```
 1: // $Id: commands.h,v 1.2 2010-12-13 20:11:09-08 - - $
 2:
 3: #ifndef __COMMANDS_H__
 4: #define __COMMANDS_H__
 5:
 6: #include <map>
 7:
 8: using namespace std;
 9:
10: #include "inode.h"
11: #include "trace.h"
12: #include "util.h"
13:
14: //
15: // A couple of convenient typedefs to avoid verbosity.
16: //
17:
18: typedef void (*function) (inode_state &state, const wordvec &words);
19: typedef map<string, function> commandmap;
20:
21: //
22: // commands -
23: //    A class to hold and dispatch each of the command functions.
24: //    Each command "foo" is interpreted by a function fn_foo.
25: // ctor -
26: //    The default ctor initializes the map.
27: // operator[] -
28: //    Given a string, returns a function associated with it,
29: //    or 0 if not found.
30: //
31:
32: class commands {
33:    private:
34:       commands (const inode &); // disable copy ctor
35:       commands &operator= (const inode &); // disable operator=
36:       commandmap map;
37:    public:
38:       commands();
39:       function operator[] (const string &cmd);
40: };
41:
42: //
43: // execution functions -
44: //    See the man page for a description of each of these functions.
45: //
46:
47: void fn_cat    (inode_state &state, const wordvec &words);
48: void fn_cd     (inode_state &state, const wordvec &words);
49: void fn_echo   (inode_state &state, const wordvec &words);
50: void fn_exit   (inode_state &state, const wordvec &words);
51: void fn_ls     (inode_state &state, const wordvec &words);
52: void fn_lsr    (inode_state &state, const wordvec &words);
53: void fn_make   (inode_state &state, const wordvec &words);
54: void fn_mkdir  (inode_state &state, const wordvec &words);
55: void fn_prompt (inode_state &state, const wordvec &words);
56: void fn_pwd    (inode_state &state, const wordvec &words);
57: void fn_rm     (inode_state &state, const wordvec &words);
58: void fn_rmr    (inode_state &state, const wordvec &words);
59:
60: //
61: // exit_status_message -
62: //    Prints an exit message and returns the exit status, as recorded
63: //    by any of the functions.
64: //
```

```
65:
66: int exit_status_message();
67: class ysh_exit_exn: public exception {};
68:
69: #endif
70:
```

```
 1: // $Id: inode.h,v 1.3 2010-12-13 20:11:09-08 - - $
 2:
 3: #ifndef __INODE_H__
 4: #define __INODE_H__
 5:
 6: #include <exception>
 7: #include <iostream>
 8: #include <map>
 9: #include <vector>
10:
11: using namespace std;
12:
13: #include "trace.h"
14: #include "util.h"
15:
16: //
17: // inode_t -
18: //    An inode is either a directory or a plain file.
19: //
20:
21: enum inode_t {DIR_INODE, FILE_INODE};
22:
23: //
24: // directory -
25: //    A directory is a list of paired strings (filenames) and inodes.
26: //    An inode in a directory may be a directory or a file.
27: //
28:
29: class inode;
30: typedef map<string, inode *> directory;
31:
32: //
33: // class inode -
34: //
35: // inode ctor -
36: //    Create a new inode of the given type, using a union.
37: //
38: // get_inode_nr -
39: //    Retrieves the serial number of the inode.  Inode numbers are
40: //    allocated in sequence by small integer.
41: //
42: // size -
43: //    Returns the size of an inode.  For a directory, this is the
44: //    number of dirents.  For a text file, the number of characters
45: //    when printed (the sum of the lengths of each word, plus the
46: //    number of words.
47: //
48: // readfile -
49: //    Returns a copy of the contents of the wordvec in the file.
50: //    Throws an yshell_exn for a directory.
51: //
52: // writefile -
53: //    Replaces the contents of a file with new contents.
54: //    Throws an yshell_exn for a directory.
55: //
56: // remove -
57: //    Removes the file or subdirectory from the current inode.
58: //    Throws an yshell_exn if this is not a directory, the file
59: //    does not exist, or the subdirectory is not empty.
60: //    Here empty means the only entries are dot (.) and dotdot (..).
61: //
62: // mkdir -
63: //    Creates a new directory under the current directory and
64: //    immediately adds the directories dot (.) and dotdot (..) to it.
```

```
 65: //    Note that the parent (..) of / is / itself.  It is an error
 66: //    if the entry already exists.
 67: //
 68: // mkfile -
 69: //    Create a new empty text file with the given name.  Error if
 70: //    a dirent with that name exists.
 71: //
 72: //
 73:
 74: class inode {
 75:    private:
 76:       int inode_nr;
 77:       inode_t type;
 78:       union {
 79:          directory *dirents;
 80:          wordvec *data;
 81:       } contents;
 82:       static int next_inode_nr;
 83:    public:
 84:       inode (inode_t init_type);
 85:       inode (const inode &source);
 86:       inode &operator= (const inode &from);
 87:       int get_inode_nr();
 88:       int size();
 89:       const wordvec &readfile() const;
 90:       void writefile (const wordvec &newdata);
 91:       void remove (const string &filename);
 92: };
 93:
 94: //
 95: // inode_state -
 96: //    A small convenient class to maintain the state of the simulated
 97: //    process:  the root (/), the current directory (.), and the
 98: //    prompt.
 99: //
100:
101: class inode_state {
102:    friend class inode;
103:    friend ostream &operator<< (ostream &out, const inode_state &);
104:    private:
105:       inode_state (const inode_state &); // disable copy ctor
106:       inode_state &operator= (const inode_state &); // disable op=
107:       inode *root;
108:       inode *cwd;
109:       string prompt;
110:    public:
111:       inode_state();
112: };
113:
114: ostream &operator<< (ostream &out, const inode_state &);
115:
116: #endif
117:
```

```
 1: // $Id: trace.h,v 1.3 2010-12-13 20:11:09-08 - - $
 2:
 3: #ifndef __TRACE_H__
 4: #define __TRACE_H__
 5:
 6: #include <string>
 7: #include <vector>
 8:
 9: using namespace std;
10:
11: //
12: // traceflags -
13: //    static class for maintaining global trace flags, each indicated
14: //    by a single character.
15: // setflags -
16: //    Takes a string argument, and sets a flag for each char in the
17: //    string.  As a special case, '@', sets all flags.
18: // getflag -
19: //    Used by the TRACE macro to check to see if a flag has been set.
20: //    Not to be called by user code.
21: //
22:
23: class traceflags {
24:    private:
25:       static vector<char> flags;
26:    public:
27:       static void setflags (const string &optflags);
28:       static bool getflag (char flag);
29: };
30:
31: //
32: // TRACE -
33: //    Macro which expands into trace code.  First argument is a
34: //    trace flag char, second argument is output code that can
35: //    be sandwiched between <<.  Beware of operator precedence.
36: //    Example:
37: //       TRACE ('u', "foo = " << foo);
38: //    will print two words and a newline if flag 'u' is  on.
39: //    Traces are preceded by filename, line number, and function.
40: //
41:
42: #define TRACE(FLAG,CODE) { \
43:            if (traceflags::getflag (FLAG)) { \
44:               cerr << __FILE__ << ":" << __LINE__ << ":" \
45:                    << __func__ << ":" << endl; \
46:               cerr << CODE << endl; \
47:            } \
48:         }
49:
50: #endif
51:
```

```
 1: // $Id: util.h,v 1.3 2010-12-13 20:11:09-08 - - $
 2:
 3: //
 4: // util -
 5: //    A utility class to provide various services not conveniently
 6: //    included in other modules.
 7: //
 8:
 9: #ifndef __UTIL_H__
10: #define __UTIL_H__
11:
12: #include <iostream>
13: #include <string>
14: #include <vector>
15:
16: #ifdef __GNUC__
17: #include <stdexcept>
18: #endif
19:
20: using namespace std;
21:
22: #include "trace.h"
23:
24: //
25: // A couple of convenient typedefs to allow brevity of code elsewhere.
26: //
27:
28: typedef vector<string> wordvec;
29: typedef wordvec::const_iterator wordvec_itor;
30:
31: //
32: // yshell_exn -
33: //    Extend runtime_error for throwing exceptions related to this
34: //    program.
35: //
36:
37: class yshell_exn: public runtime_error {
38:    public:
39:       explicit yshell_exn (const string &what);
40: };
41:
42: //
43: // setexecname -
44: //    Sets the static string to be used as an execname.
45: // execname -
46: //    Returns the basename of the executable image, which is used in
47: //    printing error messags.
48: //
49:
50: void setexecname (const string &);
51: string &execname();
52:
53: //
54: // want_echo -
55: //    We want to echo all of cin to cout if if either cin or cout
56: //    is not a tty.  This helps make batch processing easier by
57: //    making cout look like a terminal session trace.
58: //
59:
60: bool want_echo();
61:
62: //
63: // exit_status -
64: //    A static class for maintaining the exit status.  The default
```

```
 65: //    status is EXIT_SUCCESS (0), but can be set to another value,
 66: //    such as EXIT_FAILURE (1) to indicate that error messages have
 67: //    been printed.
 68: //
 69:
 70: class exit_status {
 71:    private:
 72:       static int status;
 73:    public:
 74:       static void set (int);
 75:       static int get();
 76: };
 77:
 78: //
 79: // split -
 80: //    Split a string into a wordvec (as defined above).  Any sequence
 81: //    of chars in the delimiter string is used as a separator.  To
 82: //    Split a pathname, use "/".  To split a shell command, use " ".
 83: //
 84:
 85: wordvec split (const string &line, const string &delimiter);
 86:
 87: // complain -
 88: //    Used for starting error messages.  Sets the exit status to
 89: //    EXIT_FAILURE, writes the program name to cerr, and then
 90: //    returns the cerr ostream.  Example:
 91: //       complain() << filename << ": some problem" << endl;
 92: //
 93:
 94: ostream &complain();
 95:
 96: //
 97: // operator<< (vector) -
 98: //    An overloaded template operator which allows vectors to be
 99: //    printed out as a single operator, each element separated from
100: //    the next with spaces.  The item_t must have an output operator
101: //    defined for it.
102: //
103:
104: template <typename item_t>
105: ostream &operator<< (ostream &out, const vector<item_t> &vec);
106:
107: //
108: // Put the RCS Id string in the object file.
109: //
110:
111: #endif
112:
```

```
 1: // $Id: commands.cc,v 1.3 2010-12-13 20:11:09-08 - - $
 2:
 3: #include "commands.h"
 4: #include "trace.h"
 5:
 6: commands::commands(): map (commandmap()) {
 7:    map["cat"    ] = fn_cat    ;
 8:    map["cd"     ] = fn_cd     ;
 9:    map["echo"   ] = fn_echo   ;
10:    map["exit"   ] = fn_exit   ;
11:    map["ls"     ] = fn_ls     ;
12:    map["lsr"    ] = fn_lsr    ;
13:    map["make"   ] = fn_make   ;
14:    map["mkdir"  ] = fn_mkdir  ;
15:    map["prompt" ] = fn_prompt ;
16:    map["pwd"    ] = fn_pwd    ;
17:    map["rm"     ] = fn_rm     ;
18: }
19:
20: function commands::operator[] (const string& cmd) {
21:    return map[cmd];
22: }
23:
24: void fn_cat (inode_state &state, const wordvec &words){
25:    TRACE ('c', state);
26:    TRACE ('c', words);
27: }
28:
29: void fn_cd (inode_state &state, const wordvec &words){
30:    TRACE ('c', state);
31:    TRACE ('c', words);
32: }
33:
34: void fn_echo (inode_state &state, const wordvec &words){
35:    TRACE ('c', state);
36:    TRACE ('c', words);
37: }
38:
39: void fn_exit (inode_state &state, const wordvec &words){
40:    TRACE ('c', state);
41:    TRACE ('c', words);
42:    throw ysh_exit_exn ();
43: }
44:
45: void fn_ls (inode_state &state, const wordvec &words){
46:    TRACE ('c', state);
47:    TRACE ('c', words);
48: }
49:
50: void fn_lsr (inode_state &state, const wordvec &words){
51:    TRACE ('c', state);
52:    TRACE ('c', words);
53: }
54:
55: void fn_make (inode_state &state, const wordvec &words){
56:    TRACE ('c', state);
57:    TRACE ('c', words);
58: }
59:
60: void fn_mkdir (inode_state &state, const wordvec &words){
61:    TRACE ('c', state);
62:    TRACE ('c', words);
63: }
64:
```

```
65: void fn_prompt (inode_state &state, const wordvec &words){
66:    TRACE ('c', state);
67:    TRACE ('c', words);
68: }
69:
70: void fn_pwd (inode_state &state, const wordvec &words){
71:    TRACE ('c', state);
72:    TRACE ('c', words);
73: }
74:
75: void fn_rm (inode_state &state, const wordvec &words){
76:    TRACE ('c', state);
77:    TRACE ('c', words);
78: }
79:
80: void fn_rmr (inode_state &state, const wordvec &words){
81:    TRACE ('c', state);
82:    TRACE ('c', words);
83: }
84:
85: int exit_status_message() {
86:    int exit_status = exit_status::get();
87:    cout << execname() << ": exit(" << exit_status << ")" << endl;
88:    return exit_status;
89: }
90:
```

```
 1: // $Id: inode.cc,v 1.5 2010-12-13 20:11:09-08 - - $
 2:
 3: #include <cassert>
 4: #include <iostream>
 5:
 6: using namespace std;
 7:
 8: #include "trace.h"
 9: #include "inode.h"
10:
11: int inode::next_inode_nr = 1;
12:
13: inode::inode(inode_t init_type):
14:    inode_nr (next_inode_nr++), type (init_type)
15: {
16:    switch (type) {
17:       case DIR_INODE:
18:            contents.dirents = new directory();
19:            break;
20:       case FILE_INODE:
21:            contents.data = new wordvec();
22:            break;
23:    }
24:    TRACE ('i', "inode " << inode_nr << ", type = " << type);
25: }
26:
27: //
28: // copy ctor -
29: //    Make a copy of a given inode.  This should not be used in
30: //    your program if you can avoid it, since it is expensive.
31: //    Here, we can leverage operator=.
32: //
33: inode::inode (const inode &that) {
34:    *this = that;
35: }
36:
37: //
38: // operator= -
39: //    Assignment operator.  Copy an inode.  Make a copy of a
40: //    given inode.  This should not be used in your program if
41: //    you can avoid it, since it is expensive.
42: //
43: inode &inode::operator= (const inode &that) {
44:    if (this != &that) {
45:       inode_nr = that.inode_nr;
46:       type = that.type;
47:       contents = that.contents;
48:    }
49:    TRACE ('i', "inode " << inode_nr << ", type = " << type);
50:    return *this;
51: }
52:
53: int inode::get_inode_nr() {
54:    TRACE ('i', "inode = " << inode_nr);
55:    return inode_nr;
56: }
57:
58: int inode::size() {
59:    int size = 0;
60:    TRACE ('i', "size = " << size);
61:    return size;
62: }
63:
64: const wordvec &inode::readfile() const {
```

```
65:    TRACE ('i', *contents.data);
66:    assert (type == FILE_INODE);
67:    return *contents.data;
68: }
69:
70: void inode::writefile (const wordvec &words) {
71:    TRACE ('i', words);
72:    assert (type == FILE_INODE);
73: }
74:
75: void inode::remove (const string &filename) {
76:    TRACE ('i', filename);
77:    assert (type == DIR_INODE);
78: }
79:
80: inode_state::inode_state(): root (NULL), cwd (NULL), prompt ("%") {
81:    TRACE ('i', "root = " << (void*) root << ", cwd = " << (void*) cwd
82:           << ", prompt = " << prompt);
83: }
84:
85: ostream &operator<< (ostream &out, const inode_state &state) {
86:    out << "inode_state: root = " << state.root
87:        << ", cwd = " << state.cwd;
88:    return out;
89: }
90:
```

```
 1: // $Id: trace.cc,v 1.2 2010-12-13 20:11:09-08 - - $
 2:
 3: #include <iostream>
 4: #include <climits>
 5: #include <vector>
 6:
 7: using namespace std;
 8:
 9: #include "trace.h"
10:
11: //
12: // ** BUG IN STL ** BUG IN STL **
13: // We should use vector<bool> instead of vector<char>,
14: // but vector<bool> has a bug:
15: // http://forums.sun.com/thread.jspa?threadID=5277939
16: // Static linking works, but doubles the size of the executable
17: // image.
18: // ** BUG IN STL ** BUG IN STL **
19: //
20:
21: typedef vector<char> boolvec;
22: boolvec traceflags::flags (UCHAR_MAX + 1, false);
23: const boolvec trueflags (UCHAR_MAX + 1, true);
24:
25: void traceflags::setflags (const string &optflags) {
26:    string::const_iterator itor = optflags.begin();
27:    string::const_iterator end = optflags.end();
28:    for (; itor != end; ++itor) {
29:       if (*itor == '@') {
30:          flags = trueflags;
31:       }else {
32:          flags[*itor] = true;
33:       }
34:    }
35:    // Note that TRACE can trace setflags.
36:    TRACE ('t',  "optflags = " << optflags);
37: }
38:
39: //
40: // getflag -
41: //    Check to see if a certain flag is on.
42: //
43:
44: bool traceflags::getflag (char flag) {
45:    // Bug alert:
46:    // Don't TRACE this function or the stack will blow up.
47:    bool result = flags[flag];
48:    return result;
49: }
50:
```

```
 1: // $Id: util.cc,v 1.4 2010-12-13 20:11:09-08 - - $
 2:
 3: #include <cstdlib>
 4: #include <unistd.h>
 5:
 6: using namespace std;
 7:
 8: #include "util.h"
 9:
10: yshell_exn::yshell_exn (const string &what): runtime_error (what) {
11: }
12:
13: int exit_status::status = EXIT_SUCCESS;
14: static string execname_string;
15:
16: void exit_status::set (int new_status) {
17:    status = new_status;
18: }
19:
20: int exit_status::get() {
21:    return status;
22: }
23:
24: void setexecname (const string &name) {
25:    execname_string =  name.substr (name.rfind ('/') + 1);
26:    TRACE ('u', execname_string);
27: }
28:
29: string &execname() {
30:    TRACE ('u', execname_string);
31:    return execname_string;
32: }
33:
34: bool want_echo() {
35:    const int CIN_FD = 0;
36:    const int COUT_FD = 1;
37:    bool cin_isatty = isatty (CIN_FD);
38:    bool cout_isatty = isatty (COUT_FD);
39:    TRACE ('u', "cin_isatty = " << cin_isatty
40:           << ", cout_isatty = " << cout_isatty);
41:    return ! cin_isatty || ! cout_isatty;
42: }
43:
44:
45: wordvec split (const string &line, const string &delimiters) {
46:    wordvec words;
47:    size_t end = 0;
48:
49:    // Loop over the string, splitting out words, and for each word
50:    // thus found, append it to the output wordvec.
51:    for (;;) {
52:       size_t start = line.find_first_not_of (delimiters, end);
53:       if (start == string::npos) break;
54:       end = line.find_first_of (delimiters, start);
55:       words.push_back (line.substr (start, end - start));
56:    }
57:    TRACE ('u', words);
58:    return words;
59: }
60:
61: ostream &complain() {
62:    exit_status::set (EXIT_FAILURE);
63:    cerr << execname() << ": ";
64:    return cerr;
```

```
65: }
66:
67: template <typename item_t>
68: ostream &operator<< (ostream &out, const vector<item_t> &vec) {
69:    typename vector<item_t>::const_iterator itor = vec.begin();
70:    typename vector<item_t>::const_iterator end = vec.end();
71:
72:    // If the vector is empty, do nothing.
73:    if (itor != end) {
74:       // Print out the first element without a space.
75:       out << *itor++;
76:       // Print out the rest of the elements each preceded by a space.
77:       while (itor != end) out << " " << *itor++;
78:    }
79:    return out;
80: }
81:
```

```
// $Id: yshell.cc,v 1.2 2010-12-13 20:11:09-08 - - $

#include <cstdlib>
#include <iostream>
#include <string>
#include <utility>
#include <unistd.h>

using namespace std;

#include "commands.h"
#include "trace.h"
#include "inode.h"
#include "util.h"

//
// scan_options
//    Options analysis:  The only option is -Dflags.
//

void scan_options (int argc, char **argv) {
   opterr = 0;
   for (;;) {
      int option = getopt (argc, argv, "@:");
      if (option == EOF) break;
      switch (option) {
         case '@':
            traceflags::setflags (optarg);
            break;
         default:
            complain() << "-" << (char) option << ": invalid option"
                       << endl;
            break;
      }
   }
   if (optind < argc) {
      complain() << "operands not permitted" << endl;
   }
}

//
// main -
//    Main program which loops reading commands until end of file.
//

int main (int argc, char **argv) {
   setexecname (argv[0]);
   cout << boolalpha; // Print false or true instead of 0 or 1.
   cerr << boolalpha;
   scan_options (argc, argv);
   bool need_echo = want_echo();
   commands cmdmap;
   string prompt = "%";
   inode_state state;
   try {
      for (;;) {
         try {

            // Read a line, break at EOF, and echo print the prompt
            // if one is needed.
            cout << prompt << " ";
            string line;
            getline (cin, line);
            if (cin.eof()) {
```

```
65:                if (need_echo) cout << "^D";
66:                cout << endl;
67:                TRACE ('y', "EOF");
68:                break;
69:             }
70:             if (need_echo) cout << line << endl;
71:
72:             // Split the line into words and lookup the appropriate
73:             // function.  Complain or call it.
74:             wordvec words = split (line, " \t");
75:             TRACE ('y', "words = " << words);
76:             function fn = cmdmap[words[0]];
77:             if (fn == NULL) {
78:                throw yshell_exn (words[0] + ": no such function");
79:             }
80:             fn (state, words);
81:          }catch (yshell_exn exn) {
82:             // If there is a problem discovered in any function, an
83:             // exn is thrown and printed here.
84:             complain() << exn.what() << endl;
85:          }
86:       }
87:    } catch (ysh_exit_exn) {
88:    }
89:
90:    return exit_status_message();
91: }
92:
```

```
 1: # $Id: Makefile,v 1.4 2010-12-13 20:12:44-08 - - $
 2:
 3: MKFILE      = Makefile
 4: DEPSFILE    = ${MKFILE}.deps
 5: NOINCL      = ci clean spotless
 6: NEEDINCL    = ${filter ${NOINCL}, ${MAKECMDGOALS}}
 7: GMAKE       = ${MAKE} --no-print-directory
 8: UNAME      ?= ${shell uname -s}
 9:
10: ifeq (${UNAME},SunOS)
11: COMPILECCC  = CC -g -features=extensions
12: MAKEDEPSCCC = CC -xM1
13: endif
14: ifeq (${UNAME},Linux)
15: COMPILECCC  = g++ -g -Wall -Wextra -Werror
16: MAKEDEPSCCC = g++ -MM
17: endif
18:
19: CCSOURCE    = commands.cc inode.cc trace.cc util.cc yshell.cc
20: CCHEADER    = commands.h  inode.h  trace.h  util.h
21: EXECBIN     = yshell
22: OBJECTS     = ${CCSOURCE:.cc=.o}
23: OTHERS      = ${MKFILE} README
24: ALLSOURCES  = ${CCHEADER} ${CCSOURCE} ${OTHERS}
25: LISTING     = ../asg1-shell.code.ps
26: CLASS       = cmps109-wm.w11
27: PROJECT     = asg1
28:
29: all : ${EXECBIN}
30:
31: ${EXECBIN} : ${OBJECTS}
32:         ${COMPILECCC} -o $@ ${OBJECTS}
33:         - checksource ${CCSOURCE}
34:
35: %.o : %.cc
36:         cid + $<
37:         ${COMPILECCC} -c $<
38:
39: ci : ${ALLSOURCES}
40:         - checksource ${ALLSOURCES}
41:         cid + ${ALLSOURCES}
42:
43: lis : ${ALLSOURCES}
44:         mkpspdf ${LISTING} ${ALLSOURCES} ${DEPSFILE}
45:
46: clean :
47:         - rm ${OBJECTS} ${DEPSFILE} core ${EXECBIN}.errs
48:
49: spotless : clean
50:         - rm ${EXECBIN}
51:
52: submit : ${ALLSOURCES}
53:         - checksource ${ALLSOURCES}
54:         submit ${CLASS} ${PROJECT} ${ALLSOURCES}
55:         testsubmit ${CLASS} ${PROJECT} ${ALLSOURCES}
56:
57: deps : ${CCSOURCE} ${CCHEADER}
58:         @ echo "# ${DEPSFILE} created ‘LC_TIME=C date‘" >${DEPSFILE}
59:         ${MAKEDEPSCCC} ${CCSOURCE} | sort | uniq >>${DEPSFILE}
60:
61: ${DEPSFILE} :
62:         @ touch ${DEPSFILE}
63:         ${GMAKE} deps
64:
```

```
65: again :
66:         gmake --no-print-directory spotless deps ci all lis
67:
68: ifeq (${NEEDINCL}, )
69: include ${DEPSFILE}
70: endif
71:
```

```
1: $Id: README,v 1.1 2010-12-13 19:22:12-08 - - $
```

```
1: # Makefile.deps created Mon Dec 13 20:20:20 PST 2010
2: commands.o: commands.cc commands.h inode.h trace.h util.h
3: inode.o: inode.cc trace.h inode.h util.h
4: trace.o: trace.cc trace.h
5: util.o: util.cc util.h trace.h
6: yshell.o: yshell.cc commands.h inode.h trace.h util.h
```