

《Linux 环境程序设计》大作业报告

题目： Linux 下 ls 命令的实现

学 院 物联网工程学院

专 业 计算机科学与技术

班 级 计科 1404

学 号 1030414414

学生姓名 阎覃

二〇一七年十二月

目录

一、 设计思想	1
二、 总体设计	2
2.1 数据定义	2
2.2 系统（函数）调用	2
2.3 处理流程	4
三、 详细设计（含源程序）	5
四、 运行结果与分析	11
五、 设计体会	13
参考文献	14

一、设计思想

本次大作业采用 C++ 编写，运行于 Unix/Linux 环境。通过 `opendir`、`readdir`、`closedir` 等函数来操作目录，利用 `stat` 函数来获取文件信息，完成了一个 Linux 下 `ls` 命令的程序，并能在 Unix/Linux 环境下正确地运行。

最终实现 `-R`、`-S`、`-a`、`-i`、`-l`、`-r`、`-t`、`-l` 这几个选项的功能，即递归查看子目录、按大小排序、显示隐藏文件、显示 inode、长格式、倒序显示、按修改时间排序、单列显示。

在设计上尽可能地模仿原版的 `ls` 命令，通过 `man ls` 命令^[1] 查看原版程序的手册，并且通过亲自试验加强对命令的理解。原版的 `ls` 命令有很多细节需要注意，如图1所示，输出的行数会随着终端窗口的宽度自动调整。默认的排序是按照字母顺序排列的，并且是从上到下输出的。所以在编程时要注意输出的排版。每一项最后会添加若干制表符，在终端中每一个制表符的宽度小于等于 8 个空格，保证最后的输出可以显示最长的文件名。

由于 `opendir` 函数不能解析 home 目录，所以在设计时需要考虑如何获取 home 目录的路径，替换掉查询中的 `'~'` 字符。

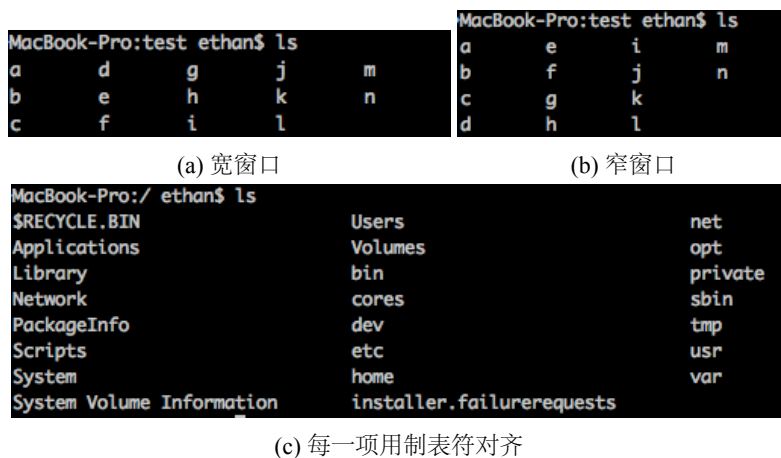


图 1: 原版的 `ls` 命令（短格式）

对于长格式的排版如图2所示，每列之间用空格隔开，用户名、用户组名、文件名是左对齐，其余右对齐，宽度要可以显示最长的一项。

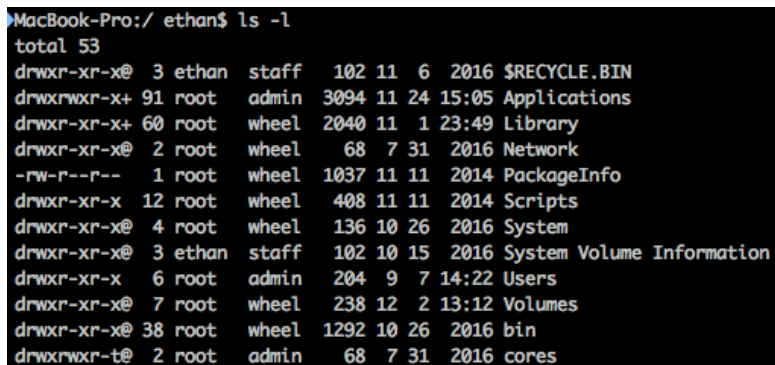


图 2: 原版的 `ls` 命令（长格式）

二、总体设计

2.1 数据定义

程序主要包含两个类：MyDir 和 MyFile。

MyDir 类表示被查询的所有目录，可以通过命令行参数传递进来的目录，也可以是递归查询的子目录。类中有两个成员变量：dir 和 name。其中 dir 是一个指向 DIR 的指针，可以通过 opendir 系统调用得到。name 是一个 string 类型的字符串，保存了目录的路径。这里的路径既可以是相对路径也可以是绝对路径，对于相对路径则始终是相对当前工作目录的路径。

MyFile 类表示被查询的所有目录，通过遍历 MyDir 得到。类中有多个成员变量，包括一个 stat 类型的成员变量 s 和多个 string 类型的输出字段。它有一个构造函数，通过输入所在文件夹的路径和文件名来初始化所有成员变量。它还重载了小于运算符，以便支持排序功能。

程序还有若干全局变量：dirs、options、homeDir、cols。

dirs 是一个存放 MyDir 的栈，它的类型是 std::stack<MyDir>。栈顶的元素就是当前需要查询的目录。当查询一个目录时，会将栈顶元素出栈。若开启了递归查询选项，则会将当前查询目录中所有文件夹类型的文件再次入栈。

options 是一个整数。利用二进制位来保存程序当前的选项。若开启某一个选项，则将对应的二进制位置 1。系统支持的选项如图3所示。

```

1 #define LONG_FORMAT      1<<0      // -l
2 #define INODE            1<<1      // -i
3 #define SORT_BY_TIME     1<<2      // -t
4 #define ALL              1<<3      // -a
5 #define ONE_PER_LINE     1<<4      // -l
6 #define SORT_BY_SIZE     1<<5      // -S
7 #define REVERSE          1<<6      // -r
8 #define RECURSIVE       1<<7      // -R

```

图 3: 配置项的二进制定义

2.2 系统（函数）调用

opendir^[2] 打开一个由文件名指定的目录，并且返回一个与其相关的目录流指针。

```

#include <dirent.h>

DIR *
opendir(const char *filename);

```

readdir^[2] 返回一个指向下一个文件的指针。

```

#include <dirent.h>

struct dirent *
readdir(DIR *dirp);

```

closedir^[2] 关闭一个目录流并且释放结构体的内存。

```

#include <dirent.h>

int
closedir(DIR *dirp);

```

getpwuid^[3] 根据提供的 uid 在用户数据库中搜索, 返回一个指向 passwd 结构体的指针, 其中包含用户名等信息。

```
#include <sys/types.h>
#include <pwd.h>
#include <uuid/uuid.h>

struct passwd *
getpwuid(uid_t uid);
```

getgrgid^[4] 根据提供的 gid 在用户组数据库中搜索, 返回一个指向 group 结构体的指针, 其中包含用户组名称等信息。

```
#include <grp.h>
#include <uuid/uuid.h>

struct group *
getgrgid(gid_t gid);
```

time^[5] 返回 1970 年 1 月 1 日 0 时 0 分 0 秒至今的毫秒数。

```
#include <time.h>

time_t
time(time_t *tloc);
```

localtime^[6] 返回一个被转换为本地时间的结构体 tm, 其中包含详细的年, 月, 日等信息。

```
#include <time.h>

struct tm *
localtime(const time_t *clock);
```

put_time^[7] 按指定的格式格式化时间。

```
#include <iomanip>

template <class charT>
/*unspecified*/ put_time (const struct tm* tmb, const charT* fmt);
```

getenv^[8] 获取环境变量

```
#include <stdlib.h>

char *
getenv(const char *name);
```

ioctl^[9] 一个专用于设备输入输出操作的系统调用, 系统调用的功能完全取决于请求码。

```
#include <sys/ioctl.h>

int
ioctl(int fildes, unsigned long request, ...);
```

2.3 处理流程

系统的流程图如图4所示。首先系统会对输入的所有参数进行处理，对于以‘-’开头的字符串，将会被识别为选项。若处理全部选项则将剩余的作为待查询目录，放入堆栈。之后对栈顶元素进行出栈，遍历此目录。若开启了递归遍历选项则将此目录中所有文件夹再次入栈。遍历完一个目录后再按照格式对所有文件进行输出。重复以上操作直到栈为空，程序结束。

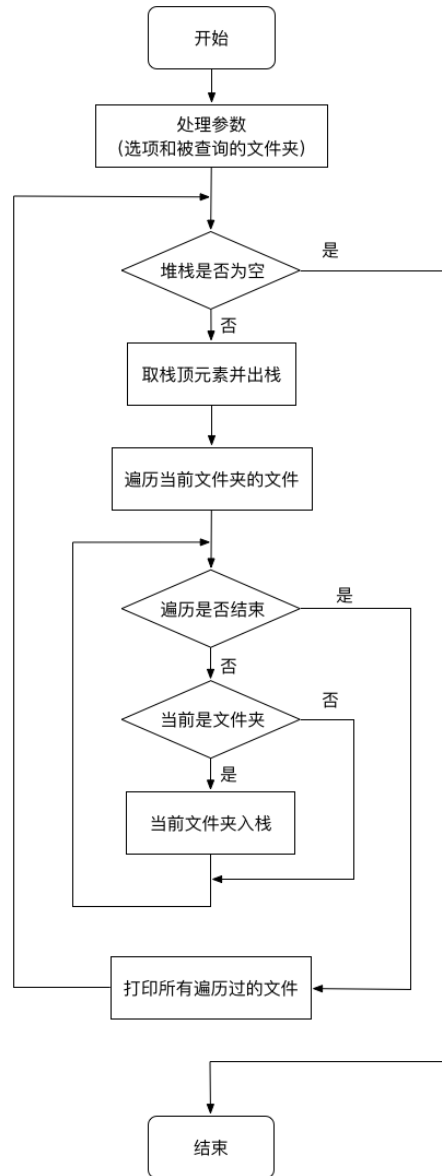


图 4: 系统流程图

三、详细设计（含源程序）

```

1  /*
2  * 简易ls程序
3  *
4  * Author: 阎覃 <ethanyt@qq.com>
5  * Blog: https://ethan-yt.github.io/
6  * Github: https://github.com/Ethan-yt/JNU-Linux-exp
7  *
8  * 本程序实现了简易的ls命令，可以用的选项有-R、-S、-a、-i、-l、
9  * -r、-t、-l，功能分别为递归查看子目录、按大小排序、显示隐藏文
10 * 件、显示inode、长格式、倒序显示、按修改时间排序、单列显示。
11 *
12 * 用法: ./ls [-RSailrtl] [file ...]
13 *
14 */
15
16 #include <iostream>
17 #include <dirent.h>
18 #include <sys/stat.h>
19 #include <sstream>
20 #include <iomanip>
21 #include <pwd.h>
22 #include <grp.h>
23 #include <unistd.h>
24 #include <sys/ioctl.h>
25 #include <vector>
26 #include <stack>
27
28 #define LONG_FORMAT      1<<0          // -l
29 #define INODE            1<<1          // -i
30 #define SORT_BY_TIME    1<<2          // -t
31 #define ALL              1<<3          // -a
32 #define ONE_PER_LINE    1<<4          // -l
33 #define SORT_BY_SIZE    1<<5          // -S
34 #define REVERSE          1<<6          // -r
35 #define RECURSIVE       1<<7          // -R
36
37 using namespace std;
38
39 struct MyDir {
40     DIR *dir;
41     string name;
42 };
43
44 stack<MyDir> dirs;          // 被查询的所有目录
45
46 const char *homeDir;      // home目录 以便替换~
47 int options = 0;          // 配置项 以二进制形式存储
48 int cols;                  // 控制台的宽度 便于格式化输出
49
50 struct MyFile {
51     struct stat s{};
52     string strInode;
53     string strLinksNumber;
54     string strOwnerName;
55     string strGroupName;
56     string strBytes;
57     string strName;
58
59     MyFile(const string &dirName, const string &fileName) {
60         string fullPath = dirName + "/" + fileName;
61         lstat(fullPath.c_str(), &s);

```

```

62
63     if (options & INODE)                                // 节点
64         strInode = to_string(s.st_ino);
65
66     if (options & LONG_FORMAT) {                          // 长格式输出
67         strLinksNumber = to_string(s.st_nlink); // 链接数
68         passwd *pw = getpwuid(s.st_uid);         // 用户名
69         strOwnerName = pw->pw_name;
70         group *gr = getgrgid(s.st_gid);          // 用户组
71         strGroupName = gr->gr_name;
72         strBytes = to_string(s.st_size);         // 字节数
73     }
74
75     strName = fileName;                                // 文件名
76 }
77
78 friend bool operator<(const MyFile &lhs, const MyFile &rhs) {
79     if (options & REVERSE) {
80         if (options & SORT_BY_SIZE && lhs.s.st_size != rhs.s.st_size)
81             return lhs.s.st_size < rhs.s.st_size;
82         if (options & SORT_BY_TIME && lhs.s.st_mtime != rhs.s.st_mtime)
83             return lhs.s.st_mtime < rhs.s.st_mtime;
84         else
85             return lhs.strName > rhs.strName;
86     } else {
87         if (options & SORT_BY_SIZE && lhs.s.st_size != rhs.s.st_size)
88             return lhs.s.st_size > rhs.s.st_size;
89         if (options & SORT_BY_TIME && lhs.s.st_mtime != rhs.s.st_mtime)
90             return lhs.s.st_mtime > rhs.s.st_mtime;
91         else
92             return lhs.strName < rhs.strName;
93     }
94 }
95
96 string getFileMode() {
97     ostream ss;
98     char c1[] = "pdcdb-ls";                                // 文件类型
99     int mask1[] = {S_IFIFO, S_IFCHR, S_IFDIR,
100                   S_IFBLK, S_IFREG, S_IFLNK, S_IFSOCK};
101     for (int n = 0; n < 7; n++) {
102         if ((s.st_mode & S_IFMT) == mask1[n]) {
103             ss << c1[n];
104             break;
105         }
106     }
107     char c2[] = "rwxrwxrwx-";                                // 属性
108     int mask2[] = {S_IRUSR, S_IWUSR, S_IXUSR, S_IRGRP,
109                   S_IWGRP, S_IXGRP, S_IROTH, S_IWOTH, S_IXOTH};
110     for (int n = 0; n < 9; n++) {
111         if (s.st_mode & mask2[n])
112             ss << c2[n];
113         else
114             ss << c2[9];
115     }
116     return ss.str();
117 }
118
119 string getDateTime() {
120     ostream ss;
121
122     time_t today;
123     time(&today);
124     tm *t = localtime(&today);                                // 今天
125     int thisYear = t->tm_year;

```



```

126         t = localtime(&s.st_mtime);           // 文件修改时间
127
128         ss << right << setw(2) << t->tm_mon + 1 << " ";
129         ss << right << setw(2) << t->tm_mday << " ";
130
131         if (t->tm_year == thisYear) {           // 今年修改 则显示时间
132             ss << put_time(t, "%H:%M");
133         } else {                                 // 非今年修改 显示年份
134             ss << put_time(t, " %Y");
135         }
136
137         return ss.str();
138     }
139
140 };
141
142 struct Info {
143     size_t maxInodeLen = 0;
144     size_t maxLinksNumberLen = 0;
145     size_t maxOwnerNameLen = 0;
146     size_t maxGroupNameLen = 0;
147     size_t maxBytesLen = 0;
148     size_t maxNameLen = 0;
149     unsigned long long blocks = 0;
150 };
151
152 // 获得最长长度以便格式化输出
153 Info getMaxLength(vector<MyFile> &files) {
154     Info ret = Info();
155     for (MyFile &f : files) {
156         if ((options & INODE) && ret.maxInodeLen < f.strInode.length())
157             ret.maxInodeLen = f.strInode.length();
158         if (options & LONG_FORMAT) {
159             if (ret.maxLinksNumberLen < f.strLinksNumber.length())
160                 ret.maxLinksNumberLen = f.strLinksNumber.length();
161             if (ret.maxOwnerNameLen < f.strOwnerName.length())
162                 ret.maxOwnerNameLen = f.strOwnerName.length();
163
164             if (ret.maxGroupNameLen < f.strGroupName.length())
165                 ret.maxGroupNameLen = f.strGroupName.length();
166             if (ret.maxBytesLen < f.strBytes.length())
167                 ret.maxBytesLen = f.strBytes.length();
168             ret.blocks += f.s.st_blocks;
169         }
170         if (ret.maxNameLen < f.strName.length())
171             ret.maxNameLen = f.strName.length();
172     }
173     ret.maxNameLen = (ret.maxNameLen / 8 + 1) * 8;
174     return ret;
175 }
176
177
178 void printUsage() {
179     cerr << "用法: ./ls [-RSailrt1] [file ...]" << endl;
180 }
181
182 void display(vector<MyFile> files, const char *dirName) {
183
184     Info info = getMaxLength(files);
185
186     sort(files.begin(), files.end());           // 排序
187
188     if (dirName != nullptr)
189         cout << dirName << ":" << endl;       // 输出当前被查

```

```

    询文件夹
190 if (options & LONG_FORMAT) { // 长格式输出
191     cout << "total " << info.blocks << endl; // total blocks
192     for (auto &file : files) {
193         if (options & INODE)
194             cout << right << setw((int) info.maxInodeLen)
195                 << file.strInode << " ";
196         cout << file.getFileMode() << " ";
197         cout << right << setw((int) info.maxLinksNumberLen)
198             << file.strLinksNumber << " ";
199         cout << left << setw((int) info.maxOwnerNameLen)
200             << file.strOwnerName << " ";
201         cout << left << setw((int) info.maxGroupNameLen)
202             << file.strGroupName << " ";
203         cout << right << setw((int) info.maxBytesLen)
204             << file.strBytes << " ";
205         cout << file.getDateTime() << " ";
206         cout << file.strName << endl;
207     }
208 } else {
209     size_t c;
210     size_t s = files.size(); // 总数
211     if (options & ONE_PER_LINE)
212         c = 1;
213     else {
214         size_t len = (options & INODE) // 每条的总长度
215                     ? info.maxNameLen + info.maxInodeLen + 1
216                     : info.maxNameLen;
217         c = cols / len; // 列数
218         if (c == 0) c = 1;
219     }
220     size_t r = s / c; // 行数
221     if (s % c > 0) r++; // 向上取整
222     for (size_t i = 0; i < r; i++) {
223         for (size_t j = 0; j < c; j++) {
224             if (j * r + i >= s) continue;
225             if (options & INODE)
226                 cout << right << setw((int) (info.maxInodeLen))
227                     << files[j * r + i].strInode << " ";
228             cout << left << setw((int) (info.maxNameLen))
229                 << files[j * r + i].strName;
230         }
231         cout << endl;
232     }
233 }
234 }
235
236 int main(int argc, char *argv[]) {
237     if ((homeDir = getenv("HOME")) == nullptr) { // 获取home目录以便替换~
238         homeDir = getpwuid(getuid())->pw_dir;
239     }
240
241     bool optionflag = true;
242     int dirNum = 0;
243
244     for (int i = 1; i < argc; i++) { // 处理参数
245         if (argv[i][0] != '-')
246             optionflag = false;
247         if (optionflag) {
248             size_t len = strlen(argv[i]);
249             for (int j = 1; j < len; j++) {
250                 switch (argv[i][j]) {
251                     case '\\':
252                         options |= LONG_FORMAT;

```

```

253         break;
254     case 'i':
255         options |= INODE;
256         break;
257     case 't':
258         options |= SORT_BY_TIME;
259         break;
260     case 'a':
261         options |= ALL;
262         break;
263     case 'l':
264         options |= ONE_PER_LINE;
265         break;
266     case 'S':
267         options |= SORT_BY_SIZE;
268         break;
269     case 'r':
270         options |= REVERSE;
271         break;
272     case 'R':
273         options |= RECURSIVE;
274         break;
275     default:
276         cerr << "ls: 非法参数 — " << argv[i][j] << endl;
277         printUsage();
278         return EPERM;
279     }
280 }
281 } else {
282     dirNum++;
283     MyDir d{};
284     string path;
285     if (argv[i][0] == '~') {
286         path = homeDir;
287         path += (argv[i] + 1);
288     } else {
289         path = argv[i];
290     }
291
292     d = {opendir(path.c_str()), path};
293     if (d.dir == nullptr)
294         cerr << "ls: " << argv[i]
295             << ": " << strerror(errno) << endl;
296     else
297         dirs.push(d);
298 }
299 }
300 }
301
302 if (dirNum == 0) {
303     MyDir d{opendir("."), "."};
304     dirs.push(d);
305     dirNum++;
306 }
307
308 if (!(options & LONG_FORMAT)) {
309     // 若不是长格式获取控制台大小
310     winsize w{};
311     ioctl(STDOUT_FILENO, TIOCGWINSZ, &w);
312     cols = (w.ws_col == 0) ? 82 : w.ws_col;
313 }
314
315 while (!dirs.empty()) {
316     // 遍历所有被查询的文件夹
317     MyDir dir = dirs.top();
318     dirs.pop();

```

```
317     vector<MyFile> files; // 存放当前目录的所有文件
318     vector<MyDir> dirfiles;
319     dirent *pDirent;
320     while ((pDirent = readdir(dir.dir))) { // 遍历每个文件夹的文件
321         if (!(options & ALL)
322             && pDirent->d_name[0] == '.') // 过滤隐藏文件
323             continue;
324         MyFile file(dir.name, pDirent->d_name);
325         if (options & RECURSIVE
326             && S_ISDIR(file.s.st_mode)
327             && file.strName != "."
328             && file.strName != "..") { // 递归查看文件夹
329             string fullPath = dir.name + "/" + file.strName;
330             DIR *d = opendir(fullPath.c_str());
331             if (d != nullptr)
332                 dirfiles.push_back({d, fullPath});
333         }
334         files.push_back(file);
335     }
336     for (auto i = dirfiles.rbegin(); i != dirfiles.rend(); i++)
337         dirs.push(*i);
338
339     closedir(dir.dir);
340     display(files, dirNum > 1 || options & RECURSIVE
341             ? dir.name.c_str()
342             : nullptr); // 输出 files
343     if (!dirs.empty()) cout << endl; // 若没有结束则加一个空行
344 }
345
346 return 0;
347 }
348 }
```

四、运行结果与分析

如图5所示，本程序的排版和 linux 系统自带的 ls 命令差别不大。

```
MacBook-Pro:cmake-build-debug ethan$ ls
1          Makefile          ls.cbp
OMakeCache.txt  cmake_install.cmake  src.cbp
OMakeFiles      ls
```

(a) 系统的 ls 命令

```
MacBook-Pro:cmake-build-debug ethan$ ./ls
1          Makefile          ls.cbp
OMakeCache.txt  cmake_install.cmake  src.cbp
OMakeFiles      ls
```

(b) 本程序

图 5: 本程序和系统的 ls 命令对比

如图6所示，当输入的文件名出错时，会有相应的错误提示。

```
MacBook-Pro:cmake-build-debug ethan$ ls abcde
ls: abcde: No such file or directory
```

(a) 系统的 ls 命令

```
MacBook-Pro:cmake-build-debug ethan$ ./ls abcde
ls: abcde: No such file or directory
```

(b) 本程序

图 6: 本程序和系统的 ls 命令对比（文件名错误）

如图7所示，当输入的选项错误时，会有用法提示。

```
MacBook-Pro:cmake-build-debug ethan$ ls -JJJJ
ls: illegal option -- J
usage: ls [-ABCFGHLOPRSTUWabcdeghiklmnopqrstuwX1] [file ...]
```

(a) 系统的 ls 命令

```
MacBook-Pro:cmake-build-debug ethan$ ./ls -JJJ
ls: 非法参数 -- J
用法: ./ls [-RSailrt1] [file ...]
```

(b) 本程序

图 7: 本程序和系统的 ls 命令对比（选项错误）

如图8所示，开启递归选项（'-R'）时，会按照深度优先的方式递归搜索所有子文件夹。

<pre>MacBook-Pro:cmake-build-debug ethan\$ ls -R 1 Makefile ls.cbp OMakeCache.txt cmake_install.cmake src.cbp OMakeFiles ls ./1: 1234567 1234567b 1234567d 1234567a 1234567c 1234567e ./OMakeFiles: 3.8.2 clion-log.txt</pre>	<pre>MacBook-Pro:cmake-build-debug ethan\$./ls -R .: 1 Makefile ls.cbp OMakeCache.txt cmake_install.cmake src.cbp OMakeFiles ls ./1: 1234567 1234567b 1234567d 1234567a 1234567c 1234567e ./OMakeFiles: 3.8.2 clion-log.txt</pre>
---	--

(a) 系统的 ls 命令

(b) 本程序

图 8: 本程序和系统的 ls 命令对比（递归搜索）

在不同宽度的窗口中，输出的行数会随之变化。图9展示了开启了选项-a、-i，即显示所有文件和 inode 时，窗口宽度不同本程序的输出。

如图10所示，输入名称中包含字符 '~' 时，会自动替换为 Home 目录的路径并成功查询。

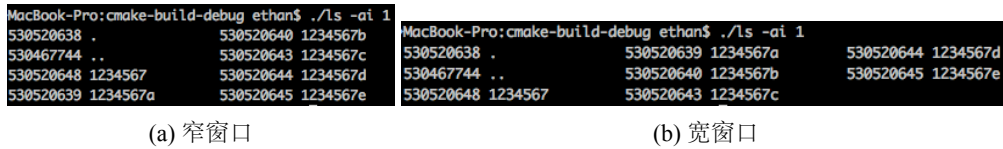


图 9: 不同宽度下的输出排版

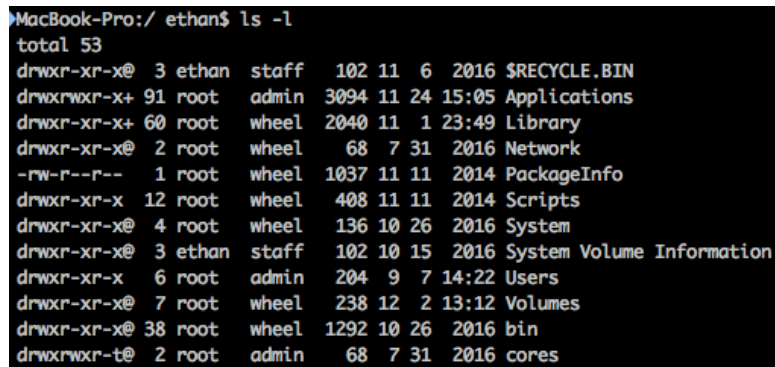


图 10: 输入字符包含 '~'

如图11所示，程序开启了-a、-i、-l、-r、-t 选项，长格式按时间升序显示所有文件，显示 inode。

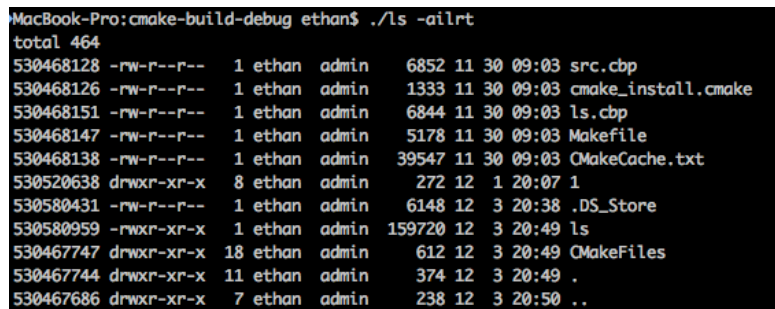


图 11: 开启选项-ailrt

如图12所示，程序开启了-l 选项，即无论窗体宽度为多少，都采用单列的方式输出。

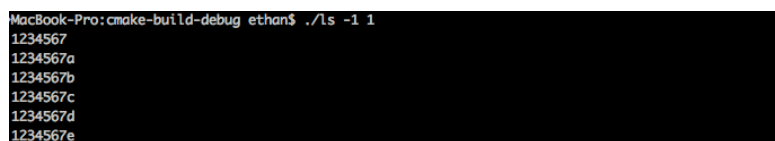


图 12: 开启选项-l

五、设计体会

经过本次的大作业，我学会了 `linux` 中关于目录和文件的一些系统调用，并且更加巩固了文件、用户、组、文件属性和许可权限的概念。在这次作业的过程中我遇到了许多问题，但是查阅相关网页资料和 `linux` 自带的手册后都可以得到解决。本次作业提高了我的实际操作能力，也提高了我编程的能力。实验报告采用 `LATEX` 排版。

参考文献

- [1] LS(1) BSD General Commands Manual. [A]. 2002.
- [2] DIRECTORY(3) BSD Library Functions Manual. [A]. 2008.
- [3] GETPWENT(3) BSD Library Functions Manual. [A]. 2011.
- [4] GETGENT(3) BSD Library Functions Manual. [A]. 2011.
- [5] TIME(3) BSD Library Functions Manual. [A]. 2003.
- [6] CTIME(3) BSD Library Functions Manual. [A]. 1999.
- [7] Put_time - C++ Reference[EB/OL]. [2017-12-3]. http://www.cplusplus.com/reference/iomanip/put_time/.
- [8] GETENV(3) BSD Library Functions Manual. [A]. 2007.
- [9] IOCTL(2) BSD System Calls Manual. [A]. 1993.