《操作系统》实验报告

实验名称： **Shell 程序设计**

姓名：肖佳伟　学号：515030910023　班级：F1503001

一. 实验目的：

进一步理解、使用和掌握 Shell 的命令调用，能利用和选择这些基本的 Shell 命令操作完成复杂的处理工作。

二. 实验内容：

1. 编写一个 Shell 程序 `findit`：

   该程序搜索参数1指定的目录树，查找所有的以 `.c` 和 `.h` 结尾的文件，如文件行中含有参数指定的字符串，显示该行和相应的文件名。如目录参数 1 缺省，则从当前目录中搜索。如：

   ```
   findit /home/wang/work searchstring
   ```

   搜索以 `/home/wang/work` 为根的目录树中的 C 程序和头文件，查找含有 `searchstring` 字符串的行，显示文件名。

   ```
   findit searchstring
   ```

   从当前目录开始搜索。

2. 编一 Shell 程序，以类似书本的目录结构的形式，按层次输出当前目录树中的所有目录和文件，要求每一层缩进 4 个空格。

三. 实验环境：

实验主机：macOS 10.13.6
实验环境：GNU bash 3.2.57

四. 算法描述：

I. 关于 `findit.sh` 脚本：

- 使用 `grep` 命令可在一行中实现本脚本：

  ```
  grep --colour=auto -n -R --include *.[ch] $pattern $cwd
  ```

  - `--colour`：颜色输出，将与 `pattern` 匹配的字符串高亮显示
  - `-n`/`--line-number`：输出文件中匹配字符串的行编号
  - `-R`/`-r`/`--recursive`：递归检索，将 `cwd` 路径中的所有文件递归匹配
  - `--include`：仅检查符合 `*.[ch]` 形式，即 C 程序或头文件的文件

- 或首先使用 `find` 命令找出符合 `*.[ch]` 形式的文件，随后使用 `grep` 命令进行匹配
  - 可使用 `grep -l`/`grep --files-with-matches` 命令检索得到含有匹配字符串的

文件名称，随后使用 grep 命令将其依此列出

- 或使用 grep -c/grep --count 命令对文件检索匹配字符串的出现次数，如其出现次数大于或等于 1，则可使用 grep 命令将其依此列出

II. 关于 tree.sh 脚本：

- 采用深度优先搜索算法：

```
DFS () {
    files <- list folder
    for file in files {
        print file
        if file is directory {
            DFC (file)
        }
    }
}
```

- 使用 Shell 中的函数递归实现上述算法，通过 (...) 创建子进程的方式使得 cd 命令不会对外界环境造成影响；同时，使用 local 关键字，确保各同名参数间的命名空间独立性
- 考虑到路径中可能存在软链接（symbolic link），为简化处理并避免闭环路径问题，使用 test -L 命令检查是否为链接文件；如是，则使用 readlink 命令找到其指向，在输出中标注，并跳过对该文件夹的递归检索

五. 测试及试验结果：

I. 脚本 findit.sh 的测试结果：

```
$ findit main
+ grep --colour=auto --line-number --recursive --include '*.[ch]' main .
./Brainfuck/utilities/qdb.c:25:int main(int argc, char **argv){
./Befunge/utilities/Befunge-93/src/bef2c.c:121:int main (int, char **);
./Befunge/utilities/Befunge-93/src/bef2c.c:125:int main (argc, argv)
./Befunge/utilities/Befunge-93/src/bef2c.c:206:  fprintf (fo, "void main ()\n{\n
signed long a; signed long b; char c; srand (time (0));\n\n");
./Befunge/utilities/Befunge-93/src/bef.c:98:          explicitly pops remaining
stack elements at end of execution.
./Befunge/utilities/Befunge-93/src/bef.c:117:            locations remain
spaces.
./Befunge/utilities/Befunge-93/src/bef.c:207:int main (int, char **);
./Befunge/utilities/Befunge-93/src/bef.c:211:int main (argc, argv)
./Befunge/utilities/Befunge-93/src/befprof.c:141:int main (int, char **);
./Befunge/utilities/Befunge-93/src/befprof.c:145:int main (argc, argv)
./C/C.c:11:int main(int argc, const char * argv[]) {
```

II. 脚本 tree.sh 的测试结果：

```
$ tree-alias ./jspcapy
./jspcapy
    .DS_Store
    .git
```

```
    [omits irrelevant files]
.gitattributes
.gitignore
LICENSE
MANIFEST.in
README.md
build
    bdist.macosx-10.13-x86_64
    lib
        jspcapy.py
deprecated
    .DS_Store
    FileError.pcap
    in0.pcap
    out
    pcap.py
    test.py
dist
    jspcapy-0.1.5.tar.gz
    jspcapy-0.2.0.tar.gz
    jspcapy-0.2.1.tar.gz
    jspcapy-0.2.2.tar.gz
    jspcapy-0.2.4.tar.gz
    jspcapy-0.2.5.tar.gz
    jspcapy-0.3.0.post1-py2.py3-none-any.whl
    jspcapy-0.3.0.post1.tar.gz
    jspcapy-0.3.0.tar.gz
    jspcapy-0.4.0-py2.py3-none-any.whl
    jspcapy-0.4.0.tar.gz
jsformat -> ../jsformat/src
jspcap -> ../jspcap/src
jspcapy.egg-info
    PKG-INFO
    SOURCES.txt
    dependency_links.txt
    entry_points.txt
    requires.txt
    top_level.txt
jspcapy.py
sample
    .DS_Store
    in.pcap
    out.json
    out.plist
    out.txt
setup.cfg
setup.py
tox.ini
```

六. 总结：

1. 事实上，在 `find` 命令中，并不存在对闭环路径的检测机制，这使得在使用该命令时需格外注意；而 `grep --recursive` 命令则存在检测机制，但其将显式报错并中止程序。因此，实验 1 的最优解法或许是自行实现一基于广度优先搜索（BFW）的程序，在搜索的过程中同时检测是否存在闭环。
2. 对于实验 2，有一他人开发的功能相似的命令 `tree`，实验脚本即是仿照其工作模式编写的 —— 将链接文件视为特殊文件，不对其进行递归搜索。其中，特别注意到链接文件的存在，如上文所述，或可自行检测闭环是否存在。

七. 源代码：

I. 脚本 findit.sh 的源代码：

```
#!/bin/bash


################################################################################
# Find files matching certain pattern.
#
# Parameter list:
#   1. File Hierarchy (Optional)
#   2. Parttern String
################################################################################


# parameter assignment
if [ -z $2 ] ; then
    cwd=.   # file hierarchy
    ptn=$1  # pattern string
else
    cwd=$1  # file hierarchy
    ptn=$2  # pattern string
fi


# file content match
set -x
grep --colour=auto --line-number --recursive --include *.[ch] $ptn $cwd
```

II. 脚本 findit.sh 的源代码：

```
#!/bin/bash


################################################################################
# List contents of directories in a tree-like format.
#
# Parameter list:
#   1. File Hierarchy (Optional)
################################################################################


# initialise macro
tab_cnt=0
if [ -z $1 ] ; then
    path=$PWD
else
    path=$1
fi


# print function
function print {
    # assign parameters
    local item=$1
    local tab_cnt=$2

    # print item
    for _ in `seq 1 $tab_cnt` ; do
```

```
        printf "    "
    done
    echo $item
}


# list contents of directories
function tree_func {
    # assign parameters
    local path=$1
    local tab_cnt=$[tab_cnt + 1]

    # create child Shell
    (
        # temporarily change directory
        cd $path

        # walk directory
        list=`ls -A . 2>/dev/null`
        for item in $list ; do
            # recursive DFS
            if [ -L $item ] ; then
                item="$item -> `readlink $item`"
                print "$item" $tab_cnt
            elif [ -d $item ] ; then
                print $item $tab_cnt
                tree_func $item
            else
                print $item $tab_cnt
            fi
        done
    )
}


# start listing
echo $path
tree_func $path
```