TinyShell 说明文档

2021 秋 程序设计基础

计11 岳章乔 2021010706

计05 聂子坤 2020010963

计16 张庆林 2021010814

计17 许继茁 2021010

目录

[零、程序的安装与执行 3](#_Toc92642632)

[0.1 Windows 平台 3](#_Toc92642633)

[0.2 Unix/Linux/Mac 平台 3](#_Toc92642634)

[一、基本功能 3](#_Toc92642635)

[1.1. 命令行框架 3](#_Toc92642636)

[1.2. Echo 指令 4](#_Toc92642637)

[1.3 Grep 指令 4](#_Toc92642638)

[1.4. Pwd 指令 5](#_Toc92642639)

[1.5. Cd 指令 5](#_Toc92642640)

[1.6. Tee 指令 6](#_Toc92642641)

[1.7. Cat 指令 6](#_Toc92642642)

[1.8 Cp 指令 7](#_Toc92642643)

[1.9. Diff 指令 7](#_Toc92642644)

[二、扩展功能 7](#_Toc92642645)

[2.1. Grep 指令 7](#_Toc92642646)

[2.2 Echo 指令 8](#_Toc92642647)

[2.3. 解释器框架 9](#_Toc92642648)

[2.4. Exit 指令 9](#_Toc92642649)

[三、异常处理 9](#_Toc92642650)

[3.1. 解释器框架 9](#_Toc92642651)

[3.2. Grep 指令 9](#_Toc92642652)

[3.3. Cd 指令 10](#_Toc92642653)

[3.4. Cp 指令 10](#_Toc92642654)

[3.5. 文件处理 10](#_Toc92642655)

[四、工程结构 11](#_Toc92642656)

[五、功能解说 12](#_Toc92642657)

[5.1. global\_var.h 12](#_Toc92642658)

[5.2. part2.h 12](#_Toc92642659)

[六、更新日志 13](#_Toc92642660)

[七、分工 14](#_Toc92642661)

[八、意见反馈 14](#_Toc92642662)

零、程序的安装与执行

在运行TinyShell 前， 在命令行输入 g++ --version，确保 GNU Compiler Collection 已经安装好。

0.1 Windows 平台

1. 打开命令提示符，把工作目录切换到 src 文件夹。

2. 在命令行输入 g++ main.cpp –o main; .\main; 编译cpp文件，然后执行。

0.2 Unix/Linux/Mac 平台

1. 打开 bash，把工作目录切换到 src 文件夹。

2. 在命令行输入 g++ main.cpp; ./a.out; 编译cpp文件，然后执行。

一、基本功能

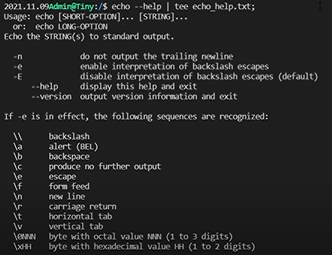
1.1. 命令行框架

命令行首先提示用户输入计算机名，根目录路径和用户名。



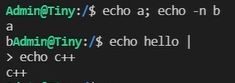
框架实现了命令行参数的标记化（tokenize）,也实现了管道处理——把 “|” 分割符左侧的输出，重定向至分隔符右侧的输入，详见样例。





1.2. Echo 指令

实现了 –n (不在结尾贴上换行符)。



从1.3 开始，测试文件都存在 examples 文件夹里。

1.3 Grep 指令

以下定义匹配行，为行内与模式串有匹配的行。

-c 输出匹配行数

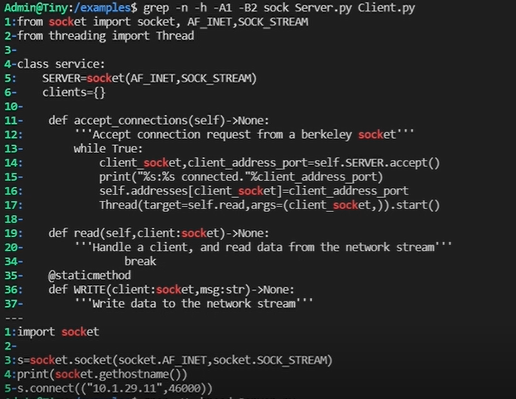


-n 显示行号（行号标为绿色，标点标为蓝色），对匹配行显示 ‘:’，非匹配行的行显示 ‘-’。

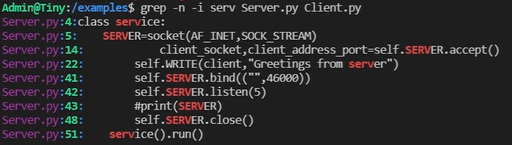
-h 强制不显示文件名（即便是多文件模式）。

-A[i] 显示匹配行后i行。

-B[i] 显示匹配行前 i 行。



-i 忽略模式串与行的大小写差异——如果不考虑大小写差异，那行与模式串匹配，那行就为匹配行。

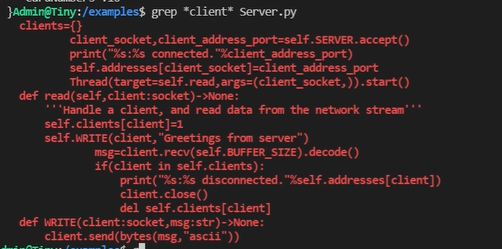


-H 强制显示文件名。



在上面的测试样例，示范了模式串中通配符’.’（一位任意字符）的实现。

以下示范通配符‘\*‘（多位任意字符）的实现。

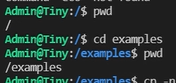


1.4. Pwd 指令

工作正常，样例见 `1.5. Cd 指令’。

1.5. Cd 指令

工作正常。



1.6. Tee 指令

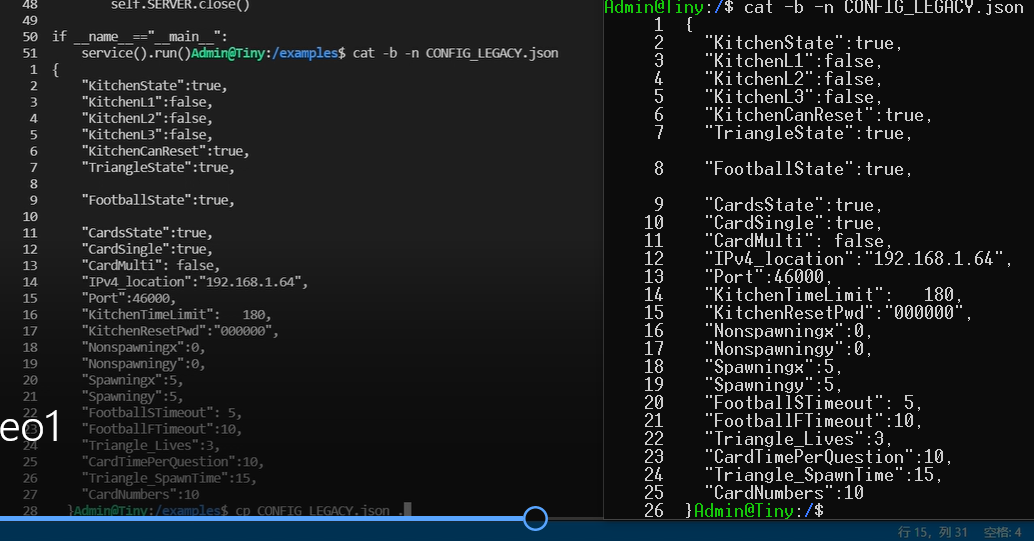
实现情况，见 1.1. 重定向的部分。

-a 如果文件不存在，就新建文件，把标准输入复制到文件里；反之就把标准输入复制在文件末尾。

1.7. Cat 指令

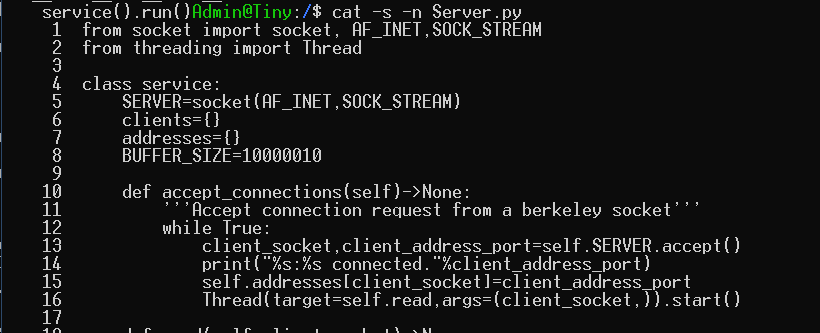
-n 输出行号。

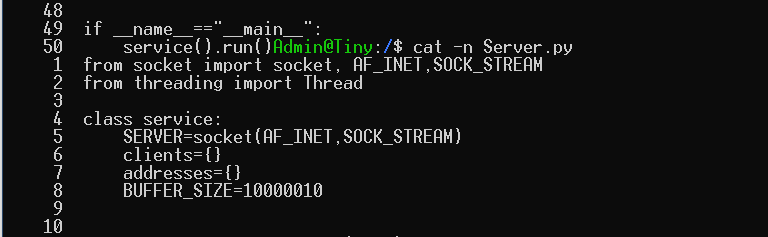
-b 对非空行输出行号，比 –n 的优先级高。



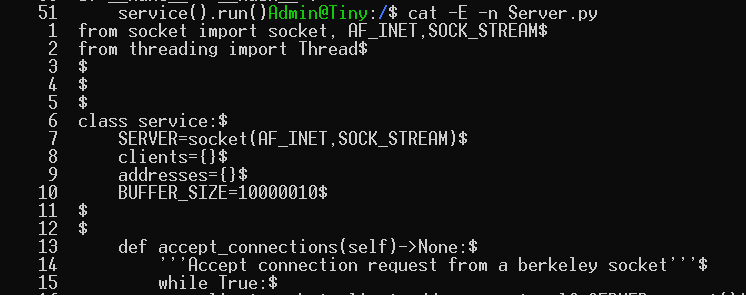
-E 在每行末尾输出‘$’ 号。

-s 输出的时候，合并连续的空行。



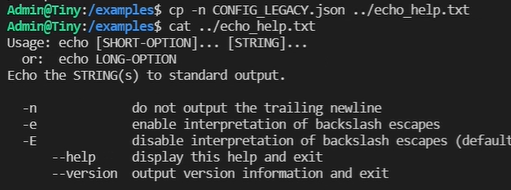






1.8 Cp 指令

-n 如果目的文件存在，就不覆盖；反之就新建文件复制源文件到那个文件。



反之，无论文件是否存在，都把那个文件内容变成是源文件的内容。

1.9. Diff 指令

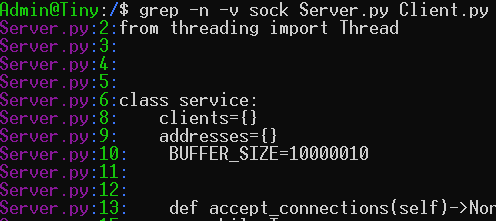
二、扩展功能

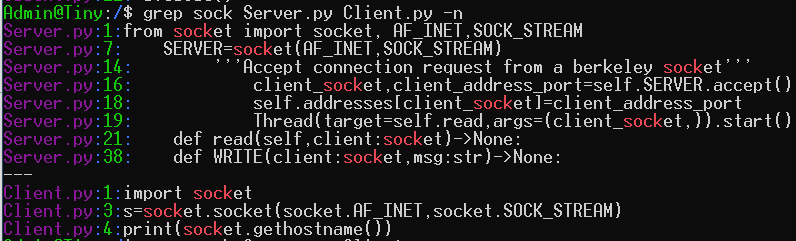
本工程的扩展功能，主要体现在解释器框架、echo指令与grep指令。

2.1. Grep 指令

2.1.1. 实现了‘-v’参数的处理，其功能为反向匹配，即输出不匹配的行。

2.1.2. 即便参数位置不同，也不影响读取。

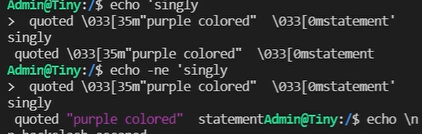




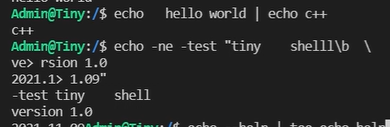
2.2 Echo 指令

-e 把命令行参数里面的特殊字符的表达式，换成相应的特殊字符。

-E -e的逆操作。



实现了把选项连着一起读的功能。



事实上，本工程的echo 与 GNU bash 的 echo 功能一致。

2.3. 解释器框架

实现了对 单引号、双引号、井号、分号的处理。

单引号： 对于引号对以内的所有字符，看作是一个行参数。

双引号： 对于引号内，除了换行符、“`”、“$” 以外，一律看作普通字符处理。

井号： 对于那一行，井号以后的所有字符都不处理，相当于注释。

分号： 表示一条命令的结尾。

另外，对标GNU Bash，如果 “|” 后面是空指令，在下一行提示用户输入指令。

展示略。

2.4. Exit 指令

退出程序。

三、异常处理

3.1. 解释器框架

对于无效的指令，输出

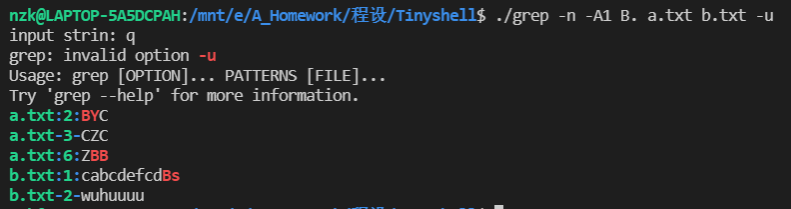
|  |
| --- |
| Command '<错误的指令>' not found |

对于程序运行期间，不涉及内存管理的例外，输出

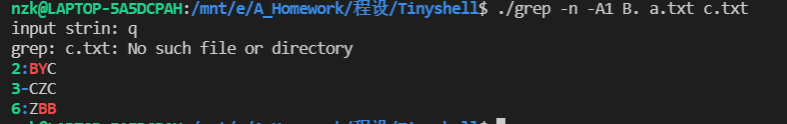
|  |
| --- |
| Error caught while running, see the message below.  <例外的解说>  Please report this bug to yuezq21@mails.tsinghua.edu.cn |

3.2. Grep 指令

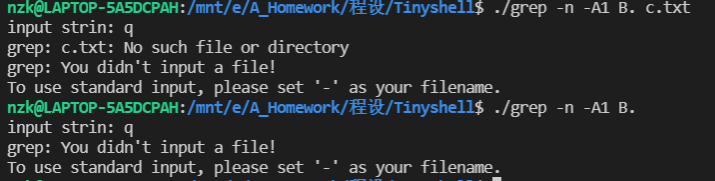
1. 参数错误：提示[invalid option]，但仍在可识别参数的基础上继续匹配。



2. 文件不存在：提示[no such file or directory]，但仍在已有文件的基础上继续匹配。



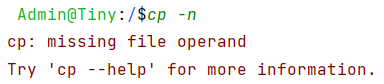
3. 无有效文件：提示使用‘-’作为标准输入。

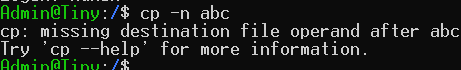


3.3. Cd 指令



3.4. Cp 指令







3.5. 文件处理

在读入文件的时候，如果在读入发现文件不存在，则提示  


四、工程结构

|  |
| --- |
| +--TinyShell\_project  |  +---- src  | |  | +- global\_var.h 存全局变量，包含 Terminal 的定义，以及共用函数  | |  | +- part1.h 模块一的头文件  | |  | +- part2.h 模块二的头文件  | |  | +- part3.h 模块三的头文件  | |  | +- part4.h 模块四的头文件  | |  | +- main.cpp 命令行所属的代码档案  | |  | +- test\_doc.txt 测试样例文件  +---- examples  | |  | +- Client.py (测试指令的文件)  | |  | +- CONFIG\_CURR.json  | |  | +- CONFIG\_LEGACY.json  | |  | +- Server.py  |  +----- documentation.docx 本文件  |  +----- documentation.pdf 本文件的 pdf 版  |  +----- Demo.mp4 TinyShell 的演示视频。 |

五、功能解说

5.1. global\_var.h

存全局变量，以及所有模块都需要用到的函数。

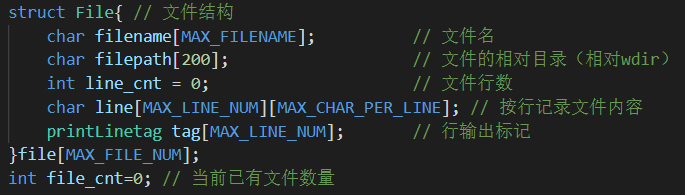
inline char\* Physical\_Path(char\* path,bool have\_root=false)；

对于各个模块间，路径定义不同的问题，在合并的时候，加入了以上的函数，无论输入是相对路径，还是绝对路径，同一输出文件的实体位置。

5.2. part2.h

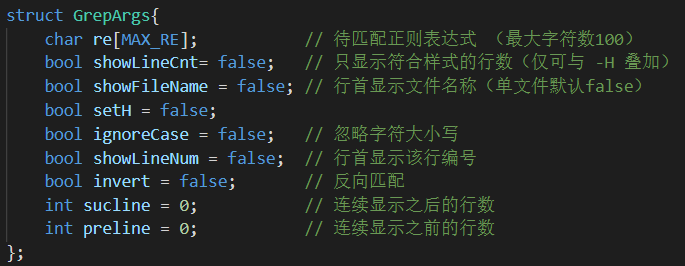
5.2.1. 数据定义

1. **File结构**



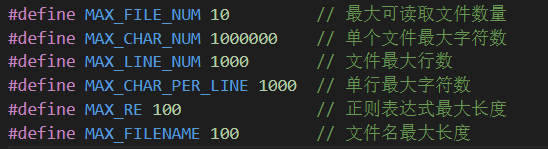
若参数为文件，每次按行读入文件的全部内容，保存在File结构数组中（标准输入stdin也作为一个名为“（standard input）”的文件记录在内）。File除记录文件内容外，还记录了控制行输出的一些标记信息。

1. **GrepArgs结构**



解析参数时记录下的参数信息，包括是否显示文件名、是否显示行号、是否忽略大小写、是否只显示行数、是否反向匹配等。doGrep函数会产生GrepArgs的一个实例，参数解析完成后将其作为实参传递给matchLine函数。

1. **处理能力约定**



5.2.2. 逻辑架构

1. 

**doGrep函数**的主体部分负责解析参数列表，完成对File和GrepArgs的设置。即读取文件信息、设置匹配参数。随后以GrepArgs的实例args作为实参传入Analyse函数。

1. 

**Analyse函数**的主体部分负责根据接收到的各种参数分析文件内容与输出匹配信息。对每个文件的每一行调用matchLine函数，该函数返回成功匹配的区间（不匹配则区间首尾为-1）。Analyse函数根据参数决定输出信息（到strout中）。

1. 

**matchLine函数**接收正则表达式、待匹配行、是否忽略大小写作为参数。从待匹配行的首位开始调用match函数匹配正则表达式。match函数返回最大匹配区间的末尾位置。一旦match函数返回有效值，matchLine即记录匹配成功的区间并返回Analyse函数。matchLine保证了匹配到的字符串是首个匹配的子串。

1. 

**match函数**接收正则表达式、待匹配子串a、是否忽略大小写作为参数，并返回最大匹配区间的末尾位置（匹配不成功则返回-1），match保证了匹配到的字符串是最长匹配的子串。

match的实现基于动态规划思想。bool dp[i][j]表示re的前i位与a的前j位是否成功匹配。若re[i]为‘.’，由于’.’可以匹配任一字符，则dp[i][j]取决于dp[i-1][j-1]是否成功；若re[i]为‘\*’，则只要dp[i-1][j]、dp[i][j-1]、dp[i-1][j-1]有一个匹配成功则dp[i][j]成功；若re[i]为普通字符，则dp[i][j]为真当且仅当dp[i-1][j-1]为真且re[i]与a[j]相同（若忽略大小写，则需二者在忽略大小写的情况下相同）。返回re全部匹配（dp的最后一行）的最后一个真即可。

设re长度为s，a长度为t，该算法**时间复杂度为O(s\*t).**

六、更新日志

七、分工

模块一：张庆林

模块二：聂子坤

模块三：许继茁

模块四：岳章乔

八、意见反馈

欢迎对程序运行出现的任何问题，包括但不限于闪退、文件读取异常等，提交反馈到 [nzk20@mails.tsinghua.edu.cn](mailto:nzk20@mails.tsinghua.edu.cn) 或 [yuezq21@mails.tsinghua.edu.cn](mailto:yuezq21@mails.tsinghua.edu.cn) 。