MyShell：跨平台简单Shell程序

基本信息

- 课程名称：Linux程序设计
- 实验项目名称：Shell程序设计
- 学生姓名：徐震
- 学号：3180105504
- 专业：计算机科学与技术
- 电子邮件地址：3180105504@zju.edu.cn
- 实验日期：2020.07.28

实验环境

硬件配置

- CPU: `2.6 GHz 6-Core Intel Core i7-9750H`
- GPU: `NVIDIA® GeForce® GTX 1650 and Intel(R) UHD Graphics 630`
- Memory: `16 GB 2666 MHz DDR4`
- Disk: `500 GB Solid State PCI-Express Drive * 2`

软件环境

- System: `Microsoft Windows 10, macOS Catalina 10.15.5 dual booting`
- Linux: `WSL2 on Windows 10, VMWare Virtual Machine Ubuntu 18.04, Manjaro USB Boot Disk, Ali Cloud ECS Server CentOS 7`
- 注意：我们会`在VMWare Virtual Machine Ubuntu 18.04上进行绝大多数实验操作（Host: Windows 10）`，如实验过程中使用了其他系统我们会注明。
- 主要实验环境详细配置：
  - 系统内核：`Linux ubuntu 5.3.0-43-generic #36~18.04.2-Ubuntu SMP Thu Mar 19 16:03:35 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux`
  - CPU：`Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz`
  - Memory：`MemTotal 6060516 kB`
- Python 3: 我们使用Python 3来实现MyShell，模拟Shell的简单功能
  - 经过测试的有：
    - `Python 3.8.2`
    - `Python 3.7.6`
    - `Python 3.6.9`
- 经过测试的系统有：
  - `Linux ubuntu 5.4.0-42-generic #46~18.04.1-Ubuntu SMP Fri Jul 10 07:21:24 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux`
  - `Linux aliecs 3.10.0-1127.13.1.el7.x86_64 #1 SMP Tue Jun 23 15:46:38 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux`

- `macOS Catalina 10.15.5`
- 系统命令有很大不同，但Shell可以基本正常运行的系统有：
- `DESKTOP-XUZH Microsoft Windows 10 Pro 10.0.18363 N/A Build 18363`
- 遗憾的是，我们没有经历测试全部的Python版本和所有可能的系统环境，但我们合理推断，在一般的 `*nix` 环境和 `Python 3` 下，MyShell都可以正常运行。在Windows环境下，MyShell的基本功能也可正常工作。

需求描述

设计文档

shell 或者命令行解释器是操作系统中最基本的用户接口。我们实现了一个跨平台的简单的shell 程序——**MyShell**，它具有以下属性：

1. 支持的内部指令集：`cd, clr, pwd, dir, echo, exit, quit, jobs, fg, bg, term, environ, set, unset, umask, printio, exec, shift, test, sleep, time, help, verbose`
   - `cd`

     更改工作目录

     `cd [target]`

     - 无参数调用时会打印当前工作目录
     - 传入一个参数调用时会尝试进入参数所示的目录
     - 在各平台上都可正常使用
     - 无法进入不存在的目录/或根本不是目录的路径/没有权限进入的路径
   - `clr`

     清空屏幕

     `clr`

     - 本指令没有参数
     - 本指令需要调用系统相关命令以管理终端屏幕
   - `pwd`

     打印当前工作目录

     `pwd [-a]`

     - 无参数调用时会打印当前工作目录，用户根目录以 `~` 显示
     - 传入参数 `-a` 调用时会打印当前工作目录完整路径
   - `dir`

     列举文件夹内容

     `dir [target [target ...]]`

     - 无参数调用时会显示当前目录下的文件列表
     - 传入多个目录时会依次显示目录的列举结果，结果中多个目录间以空行分隔
     - 对于每个目录，结果的第一行是下面将要现实的目录路径
     - 普通文件加粗显示，可执行文件以红色粗体显示，目录以蓝色粗体显示
     - 目录中的文件列表前以 `rwxrwxrwx` 格式显示文件/目录权限
     - 目录中的文件列表前显示的时间是最近修改时间
     - 若用户参数中有无法显示的目录（不存在/非目录/无权限等），会导致程序运行错误，此时将无法使用管道，但我们会打印出可以显示的那些目录的内容。
   - `echo`

打印内容

`echo [-r] [content [content ...]]`

- 无参数调用时会打印空字符串
- 传入多个参数（除了开头的 `-r` ）时会用空格分隔它们，并打印
- 传入的参数可以通过双引号包裹，被包裹的内容被视为一个整体
- 参数中可以包含 `~` 字符，会被替换为用户的根目录
- 参数中可以包含 `$...` 代表的变量，会被替换为相应的变量值，变量不存在时替换为空字符串
- 引号可以用于区分变量和普通内容，例如 `echo PATH_TO_SHELL"$SHELL"SOME_STRING` 只有一个参数，但是变量 `$SHELL` 会被正确处理
- 若要打印 `$` 符号，请输入 `\$` 以转义
- 若要打印 `~` 符号，请输入 `\~` 以转义
- 不采用 `-r` 开关时，会尝试转义传入字符中的可转义内容，例如调用 `echo "\033[1m\033[31mHello, world.\033[0m"` 会以红色粗体打印 `Hello, world.`
- 加入 `-r` 参数后，上面的命令会以普通字体打印 `\033[1m\033[31mHello, world.\033[0m`

- `exit`

退出MyShell

`exit`

- 我们不会处理任何参数，因为MyShell是一个Python Object，所以没有系统返回值的概念
- 通过调用 `exit/quit/EOF` 退出是最安全的退出方式，因为这种情况下MyShell会有机会清空还没有结束的后台工作

- `quit`

同 `exit`

- `jobs`

打印当前任务信息

`jobs`

- 我们不会处理任何参数
- 后台任务的格式为 `[i] status env command` 例如 `[0] suspended env dummy &`
- 已经被清除/已经完成的任务不会被显示
- 尝试读取内容的外部后台程序会直接获得EOF
- 任务信息是管理性质的信息，所以我们会忽略 `exec` 命令的设置，将任务管理结果直接打印到屏幕上

- `fg`

将后台任务提到前台执行

`fg job_number`

- 只接受一个参数
- 对于正在执行的后台任务，提到前台运行
- 通过外部命令的刷出的后台任务仍然不能获取输入，尝试读取内容的外部后台程序会直接获得EOF
- 对于因为获取输入而暂停执行的命令，继续命令的执行并阻塞前台主线程

- `bg`

继续后台程序的执行

`bg [job_number [job_number ...]]`

- 由于所有的暂停的后台任务都是因为尝试获取用户输入，继续在后台执行它们只会得到继续暂停的结果
- MyShell没有对快捷键操作进行处理，因此没有暂停正在运行的外部命令的功能

- `term`

  终止后台任务的执行

  `term [job_number [job_number ...]]`

  - 对于后台任务进程（`multiprocessing.Process`），发出`SIGTERM`信号以终止运行；后台任务会自动处理信号并终止自身运行
  - 若后台任务不是内部命令，会对其子进程发出`SIGKILL`信号以尝试终止运行

- `environ`

  打印MyShell全部内部变量

  `environ`

  - MyShell使用了内部的变量处理机制，在系统环境变量上加了一层额外的接口用以满足更严苛的测试环境。
  - `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`是MyShell的保留变量，不能被修改和删除

- `set`

  修改环境变量/设置新的环境变量

  `set key=value [key=value ...]`

  - 键值对以等于号配对，等于号的周围不允许出现空格，否则无法正常赋值
  - `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`是MyShell的保留变量，不能被修改（它们实际上也不存在）
  - 修改`PS1`变量会导致命令提示符的提示符号被修改，其默认值为`$`美元符号
  - 修改`PWD`等不会导致当前目录发生改变，但调用`cd`命令进入别的命令后`PWD`变量就会被修改到目录改变后的地址下
  - 修改`HOME`变量会导致程序处理`~`的方式发生改变
  - 修改`USER`等变量不会对命令提示符样式有影响，但可能会对其他使用到这些变量的程序有影响
  - 修改`PATH`可以改变程序搜索可执行文件的路径

- `unset`

  删除环境变量

  `unset key [key ...]`

  - `0, 1, 2, 3, 4, 5, 6, 7, 8, 9`是MyShell的保留变量，不能被删除（它们实际上也不存在）
  - 删除`PS1`变量会导致命令提示符采用默认值`$`
  - 删除`HOME`变量会导致程序无法正确处理`~`
  - 删除`USER`等变量不会对命令提示符样式有影响，但可能会对其他使用到这些变量的程序有影响

- `umask`

  修改程序的`umask`值

  `umask [value]`

  - 在Windows上修改`umask`的效果较为奇怪
  - 不传入参数的时候会显示当前的`umask`
  - 传入新的`umask`会被尝试以八进制解释，并设置为新的`umask`值
  - Linux上普通文本文件的默认权限是`0o666`，可执行文件为`0o777`
  - 在MyShell修改的`umask`值会影响其后的文件创建

- `printio`

  打印当前的输入输出重定向目标

  `printio`

  - 本命令没有参数
  - 本命令会打印当前MyShell的 `exec` 指令重定向目标
  - 例如执行 `exec < dummy.mysh > result.out` 后调用 `printio` 会打印

    ```
    FILE NUMBER OF INPUT FILE: 3, redirecting MyShell input to <_io.TextIOWrapper name='dummy.mysh' mode='r' encoding='utf-8'>
    FILE NUMBER OF INPUT FILE: 4, redirecting MyShell output to <_io.TextIOWrapper name='result.out' mode='w' encoding='utf-8'>
    ```

  - 由于 `printio` 的意义就在于查看当前的重定向路径，我们不会将其输入输出重定向，而是直接打印到屏幕上

- `exec`

  调整Shell的默认输入输出源

  `exec [< input] [> output | >> output]`

  - 调用本函数的效果是：若 `exec < input > output` 类似于在下面执行的每一条指令后都调用 `programname < input > output` 。但对于输出文件，输出的内容会被累积，而非像调用 `> output` 那样完全覆盖
  - 单独调用 `exec` 不会对输入输出产生任何影响，仅仅会调用 `printio` 检测当前IO状态
  - 若在某次调用中只有使用 `[< input] [> output]` 的其中之一，另一个不会被改变
  - 用户可以通过 `< ""` （传入空字符串）来清空输入源头，同样的，也可以用此方法清空输出源
  - 值得注意的是，在手册中标注不会受到 `exec` 影响（保证打印到 `sys.__stdout__` ）的程序总是会打印到 `sys.__stdout__` ，例如任务管理工作 `jobs` 或者 `exec,` `printio` 本身等。
  - 若使用的是 `>>` 符号，则会在原有的文件内容基础上添加新的输出内容。

- `shift`

  管理特殊环境变量 `1 ... 9` ，移动变量的位置（管理命令行参数）

  `shift [shamt]`

  - 通过 `$0 ... $9` 可以访问脚本/程序执行时候的命令行参数
  - `$0` 存储的是当前脚本的路径（脚本模式）/当前MyShell的路径（交互模式）
  - `$0 ... $9` 不可以被修改/删除（他们实际上也不存在于环境变量中）
  - 不带参数时，本命令可以让 `$1 ... $9` 获取下一个命令行参数，例如 `$1` 会获取 `$2` 的旧值
  - 带参数时，移动一定的数量，例如传入参数1的效果与不传入相同，传入2会使得 `$1` 获得 `$3` 的原始值
  - 调用 `shift` 命令不会修改 `$0` 的值
  - 用户的参数必须要能够转换成整数类型

- `test`

  测试表达式结果是否为真或假

  `test expression`

  - 支持的表达式：

    `-o` ：双目，逻辑或，参数为布尔值

    `-a` ：双目，逻辑与，参数为布尔值

`!` ：单目，逻辑反，参数为布尔值

`-z` ：单目，字符串长度零检查，参数为字符串

`-n` ：单目，字符串长度非零检查，参数为字符串

`==` ：双目，字符串相等性检擦，参数为字符串

`≠` ：双目，字符出不等性检查，参数为字符串

`-eq` ：双目，数值相等性检查，参数为浮点数/整数

`-ne` ：双目，数值不等性检查，参数为浮点数/整数

`-gt` ：双目，数值大于性检查 `lhs > rhs` ，参数为浮点数/整数

`-lt` ：双目，数值小于性检查 `lhs < rhs` ，参数为浮点数/整数

`-ge` ：双目，数值大于等于检查 `lhs ≥ rhs` ，参数为浮点数/整数

`-le` ：双目，数值小于等于检查 `lhs ≤ rhs` ，参数为浮点数/整数

`(` ：左括号：被括号包裹的内容会被当成一个表达式来解释，返回布尔值

`)` ：右括号：被括号包裹的内容会被当成一个表达式来解释，返回布尔值

- 表达式和运算子必须用空白符分隔开
- 支持复杂的嵌套表达式，括号/单目运算符/双目运算符皆可嵌套执行

```
test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False, -a -o from right to left
test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # True
```

- `-a, -o` 从右向左结合，但用户可以通过括号来定制它们的运算顺序
- 用户需要保证输入的内容是合理的可匹配的表达式
  - 括号需匹配完整
  - 运算符能处理的数据类型需要进行合理判断。所有经过运算的表达式结果：布尔值

- `sleep`

  等待一定时间

  `sleep amount`

  - 在*nix系统下，会尝试调用系统 `sleep` 指令，能够识别很多不同类型的睡眠时长
  - 在Windows下，会尝试调用Python 3的 `time.sleep` ，支持以秒为单位的睡眠请求

- `time`

  获取当前系统时间

  `time`

  - 以格式 `"%Y-%m-%d %H:%M:%S.%f"` 打印时间

- `help`

  获取在线帮助信息，通过 `more/less` 指令过滤

  `help [command]`

  - 无参数时，打印MyShell用户文档
  - 有参数时，打印相关指令的帮助文档，找不到MyShell内部文档时候会尝试调用系统的 `man` 指令

- `verbose`

  调整MyShell的调试信息等级

`verbose [-e|-w|-i|-d]`

- MyShell的默认调试等级为：`DEBUG`，会打印程序运行和指令执行中的最详细信息
- 推荐的日常运行等级为：`WARNING`，也就是调用`-w`后的结果
- 无参数调用时会打印当前调试等级
- 有参数调用时会尝试切换调试等级
- 也可以在启动MyShell时传入类似格式的命令行参数来修改调试信息等级

2. 开发者调试指令集：`dummy, check_zombie, queues`，用户一般不需要调用这些指令
   - `dummy`

     输入输出测试程序

     `dummy`

     - 用于检查输入请求下的后台程序暂停是否被正常实现
     - 用户可以调用一下这个指令，看看是作什么用的
   - `check_zombie`

     检查僵尸线程状态，打印`daemon`下等待主进程退出的进程

     `check_zombie`

     - 正常情况下，被手动终止的后台任务会出现在这里
     - 类似的，这类任务管理指令会被直接输出到`sys.__stdout__`
   - `queues`

     检查程序内部任务管理器的输入队列状态

     `queues`

     - 打印当前的后台任务输入队列生存状态
     - 是开发者用于检查内存泄露的方式之一
3. MyShell在开始执行后会将环境变量`SHELL`设置为`MyShell`的运行位置
4. 其他的命令行输入被解释为程序调用，MyShell创建并执行这个程序，并作为自己的子进程。程序的执行的环境变量包含一下条目：

   `PARENT=<pathname>/MyShell.py`（也就是MyShell中`SHELL`变量的内容）。
5. MyShell能够从文件中提取命令行输入，例如shell 使用以下命令行被调用：

```
./MyShell.py dummy.mysh
```

   这个批处理文件应该包含一组命令集，当到达文件结尾时MyShell退出。很明显，如果MyShell被调用时没有使用参数，它会在屏幕上显示提示符请求用户输入。
6. MyShell除了上述的脚本执行，还支持其他运行时命令行参数：

   用户可以调用`./MyShell.py -h`查看相关内容

```
usage: MyShell.py [-h] [-a [A [A ...]]] [-e] [-w] [-i] [-d] [F]

MyShell by xudenden@gmail.com

positional arguments:
  F               the batch file to be executed

optional arguments:
  -h, --help      show this help message and exit
  -a [A [A ...]]  command line arguments to batch file
  -e              enable error level debugging info log
  -w              enable warning level debugging info log
  -i              enable info level debugging info log
```

```
14      -d              enable debug(verbose) level debugging info log
```

MyShell的调用举例：

```
1    ./MyShell.py -w dummy.mysh -a foo bar foobar hello world linux linus PyTorch
    CS231n
```

7. MyShell支持I/O 重定向，stdin 和stdout，或者其中之一，例如命令行为：

```
1    programname arg1 arg2 < inputfile > outputfile
```

使用 `arg1` 和 `arg2` 执行程序 `programname` ，输入文件流被替换为 `inputfile` ，输出文件流被替换为 `outputfile` 。

`stdout` 重定向持除了在上面注明需要打印信息（后台任务管理，输入输出重定向查看等）的所有内部指令。

使用输出重定向时，如果重定向字符是 `>` ，则创建输出文件，如果存在则覆盖之；如果重定向字符为 `>>` ，也会创建输出文件，如果存在则添加到文件尾。

对于 `exec` 指令，使用重定向符号会导致MyShell的输入输出被调整到指定的文件。

MyShell在处理外部程序的调用时，为了方便用户观察结果和控制输入输出，会将 `stderr` 重定向到 `stdout` 一并打印到屏幕/定义的输出文件流。

8. MyShell支持后台程序执行。如果在命令行后添加 `&` 字符，在加载完程序后需要立刻返回命令行提示符。

   后台程序的主要管理接口为 `jobs, term, fg, bg`

   值得注意的是，通过 `subprocess` 调用的外部后台程序的输入端口是关闭的

   内部指令的输入请求会触发后台任务的暂停操作

   MyShell退出时会尝试清空所有正在运行的后台任务

9. MyShell支持管道（“|”）操作。

   在MyShell中管道和输出重定向可以同时使用而不冲突

   但输入管道和输入重定向不可同时使用

   使用管道的指令举例（请保证 `sha256sum` 指令是可用的）：

```
1    cat < dummy.mysh | wc > /dev/tty | echo "zy" > result.out | sha256sum | tr -
    d " -" >> result.out | wc | cat result.out | wc | cat result.out
```

应该会打印类似如下的内容

```
1        159     561    2940
2    zy
3    49aabdaa1b0f6c3506f54521ef81fe5b5fe835d268f1f86e1021a342b59d43bc
```

10. MyShell的命令提示符包含以下内容：

```
1    ($CONDA_DEFAULT_ENV) $USER@location $PWD time("%H:%M:%S") $PS1
```

分别为：

- 括号内的Anaconda环境
- 用户名和登陆位置名
- 当前路径（用~替换 `$HOME` 的内容）
- 当前时间（时:分:秒）
- 命令提示符符号

11. MyShell支持详细的调试信息打印，详见 `verbose` 命令的帮助手册

    一般来说我们有四种类型的信息打印：

    1. `DEBUG` 调试信息：非开发者可以忽略的调试信息，用于监测MyShell内部运行状态
    2. `INFO` 一般信息：一般性的记录信息，大部分情况下可以忽略
    3. `WARNING` 警告信息：一般在警告中出现，子进程非零退出，进程管理以及找不到的环境变量等
    4. `ERROR` 错误信息：指令格式/运行时错误

    可以在开启MyShell时通过传入命令行参数开关 `-e, -w, -i, -d` 来调整等级。

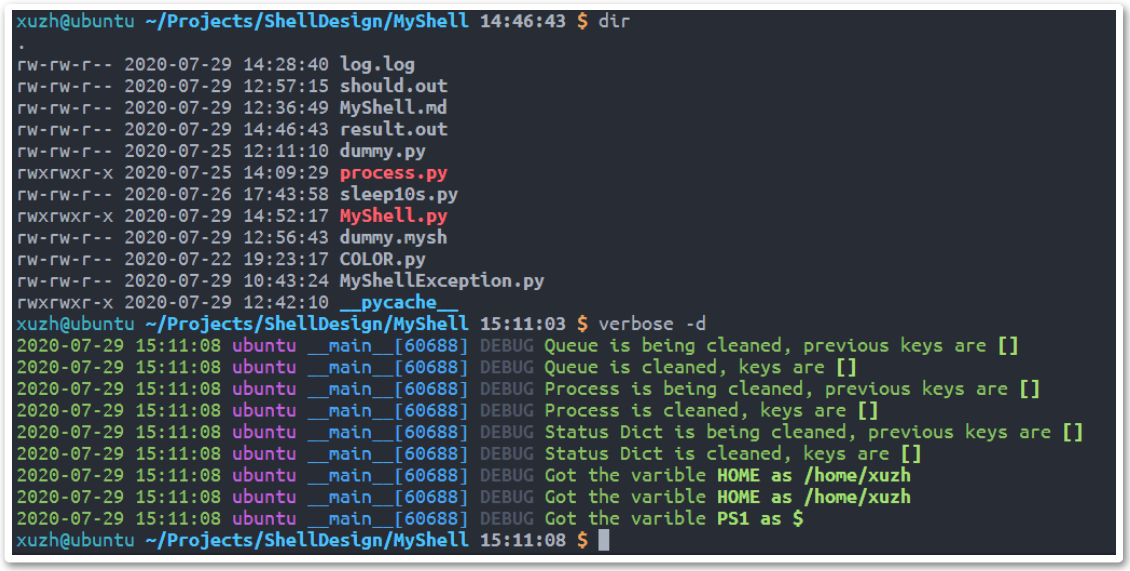
    也可以在MyShell运行时通过调用 `verbose` 指令来实现

12. MyShell支持颜色/字体调整，我们会调整输出颜色等，使其尽量容易辨识，做到用户友好

    例如在命令提示符中，我们会用不同的颜色/字体区分提示符的不同部分

    值得注意的是，用户的终端需要支持颜色输出才能正常显示相关字符，否则会有难以预料的输出错误

    我们的测试基本都是在Visual Studio Code通过SSH连接Ubuntu下执行的颜色信息的显示较为友好



13. MyShell支持定制化指令：我们使用Python实现MyShell。并在内部指令中做了统一的接口

```python
    def builtin_foo(self, pipe="", args=[]):
        # do something
        # print things that doesn't go to pipe
        # print to sys.__stdout__ to always print to STDOUT
        # return strings that go into the pipe
        return result
```

    用户只需要定义新的以 `builtin_` （注意下划线）开头的MyShell方法（包含 `pipe` 和 `args` 参数）即可添加内部指令，并无缝融入程序的运行中。

    例如：

```python
    def builtin_dummy(self, pipe="", args=[]):
        # 一个内置的dummy命令，用于测试是否可以正常触发suspension
        print("builtin_dummy: before any input requirements")
        print(input("dummy1> "))
        print(input("dummy2> "))
        print(input("dummy3> "))
        print(input("dummyend> "))
        result = input("dummy_content> ")
        return result
```

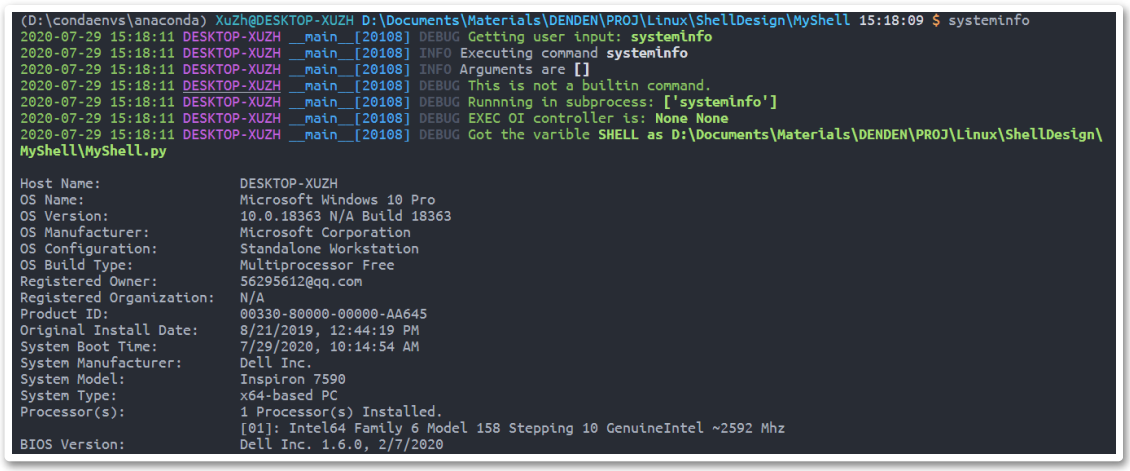
14. MyShell本体可以跨平台运行，后台任务的主体功能也可以在Windows上运行。

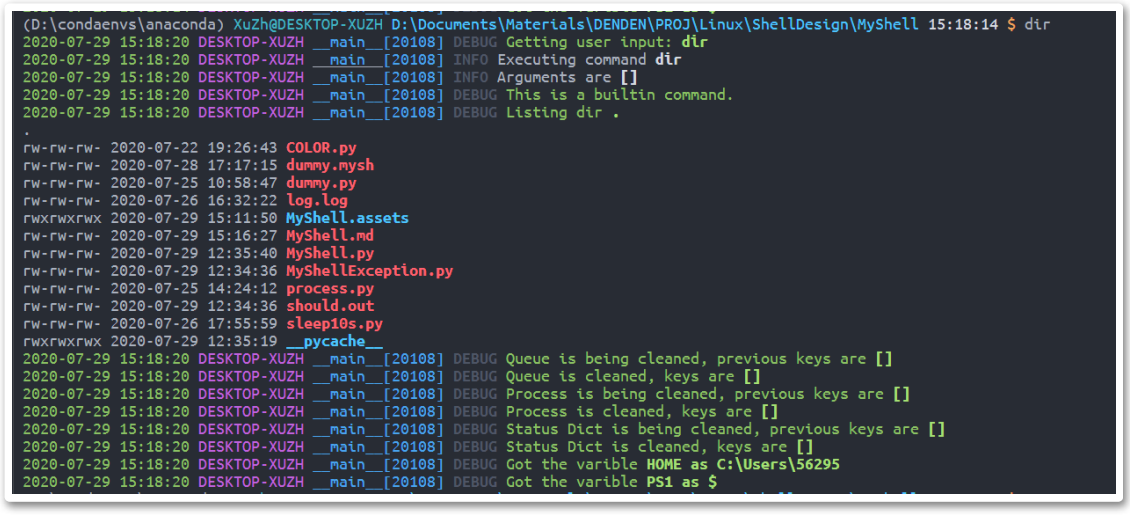
    可以运行的环境信息在本报告的前半部分已经列举过

    在第一次运行时Python 3或许会抱怨有一些包找不到，此时请通过 `pip` 来安装相关缺失的内容

    若 `pip` 速度过慢，用户可以使用 清华源 来提速

    MyShell在Windows上运行的情况：





15. MyShell有较为完备的内置报错系统

    在用户调用的命令出错时，我们会通过 `exception` 机制快速定位错误源头，并通过 `logging` 模块以人性化的方式打印相关信息

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:12 $ di
2020-07-30 11:38:13 ubuntu __main__[65482] ERROR Cannot successfully execute command "di". Exception is:
FileNotFoundException: [Errno 2] No such file or directory: 'di': 'di'
Extra info: {'type': 'subshell'}
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:13 $ cd ...
2020-07-30 11:38:22 ubuntu __main__[65482] ERROR Cannot successfully execute command "cd". Exception is:
FileNotFoundException: [Errno 2] No such file or directory: '...'
Extra info: {'type': 'cd'}
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:22 $ test ttt
True
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:28 $ test ! ttt
False
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:33 $ test ! ! ttt
True
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:38 $ test ! ( ttt
2020-07-30 11:38:41 ubuntu __main__[65482] ERROR Cannot successfully execute command "test". Exception is:
TestException: Unrecognized test expression, check your syntax. list index out of range
Extra info: None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:41 $ set 0=0
2020-07-30 11:38:53 ubuntu __main__[65482] ERROR Cannot successfully execute command "set". Exception is:
ReservedKeyException: Key 0 is reserved, along with "['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']"
Extra info: None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 11:38:53 $
```

16. MyShell支持以 `#` 开头的注释，注意 注释符号前必须是空白字符

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:31:05 $ # This is some comment
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:33:20 $ echo This is not a comment#
This is not a comment#
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:33:35 $ echo This is a comment # I'm comment!
This is a comment
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:34:11 $
```

17. MyShell支持输入历史记录的记录，用户可以通过上下方向键访问他们上次输入的内容。

    用户的左右方向键也可以被成功识别为移动光标的请求。

用户手册

程序IO重定向

Linux中的命令行程序以输入输出为主要信息交互方式，shell是Linux系列系统的基本接口之一，用户经常需要在各种各样的shell下运行不同的程序，并观察他们的输入输出结果。因此控制输入输出是shell的基本功能之一。

一般情况下，程序会从连接到终端的键盘设备（ `stdin: /dev/tty` ）读取用户的输入内容，并将输出内容打印到终端的屏幕上（ `stdout: /dev/tty` ）。

但若用户并不希望某个在shell环境下运行的程序从标准输入中 `stdin: /dev/tty` 读入内容，它可以通过 输入重定向符号 `<` 来改变shell下程序的输入源。类似的，shell也提供 输出重定向 功能。一般的，若程序以 `programname args < inputfile > outputfile` 的形式被调用，它会从 `inputfile` 中读取内容，并将标准输出导入到 `outputfile` 中。

对于输入文件流，不同于通过键盘读取的标准输出，在 `inputfile` 的内容被读完时，程序将会获得 `EOF` 信号，而非被停止并等待输入。

对于输出文件流，若使用的重定向符号为 `>` ，则会创建新的 `outputfile` 文件/覆盖原有内容。若为 `>>` ，则在 `outputfile` 已经存在或有文件内容的情况下，会在保留原有文件的基础上在文件末尾添加新的内容。

程序管道

正如上面所说的，管道其实也是输入输出重定向的一种。

不同于引导输入输出到文件，程序管道直接将上一个程序的输出作为本程序的输入，而本程序的输出会被看作下一个程序的输出。

类似的，输入内容耗尽时会获得 `EOF` 。

相对于通过文件进行交互，程序管道无需显式地对文件进行操作：这意味着其运行速度会快于通过重定向到文件。

`prog1 | prog2 | prog3` 的调用效果相当于：

```
prog1 > file1
prog2 < file1 > file2
prog3 < file2
```

程序的运行环境

在操作系统中，程序的运行环境也是控制程序运行方式的一种重要方式。

例如，我们熟悉的 `PATH` 环境变量就可以指导shell到相应的文件夹中寻找可执行文件来运行。

在一些深度学习环境中， `CUDA` 相关环境变量可以控制相应程序对Nvidia CUDA的操作方式。

`HOME` 环境变量还控制着shell对 `~` 符号的解释。

在Linux相关的shell脚本中，这些环境变量还被当作一般的变量来使用。例如我们可以将一些特殊的颜色字符储存到一个环境变量中，在以后调用相关程序需要打印相关颜色时，可以直接使用 `$COLOR`

后台程序执行

许多Linux Shell支持基于任务管理的多线程程序执行功能。我们可以通过在程序命令行末尾添加 `&` 来让程序在后台执行（特别是一些需要较长时间才能完成的程序），而立刻返回到可交互的命令行来输入其他命令。在程序完成后/状态发生改变时在shell中以一定的方式提示用户。

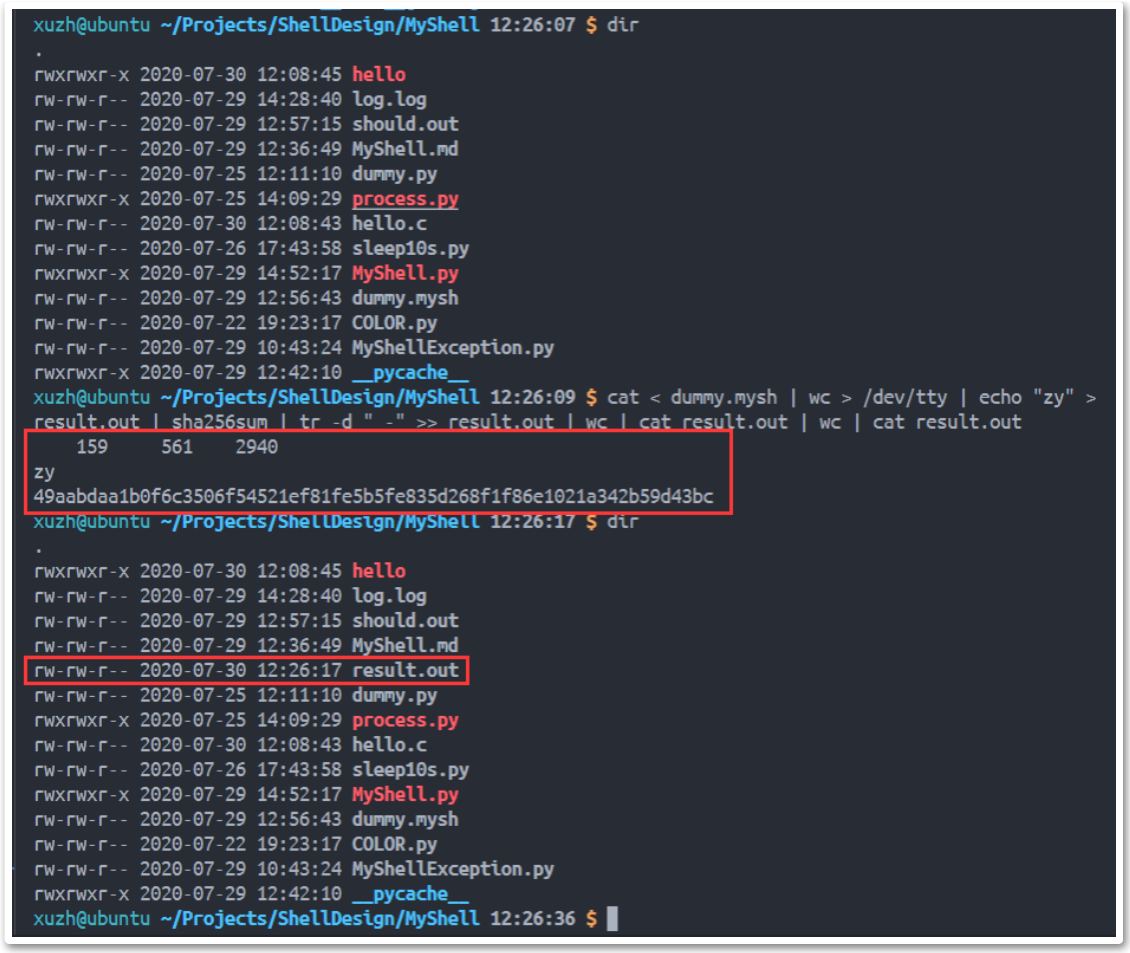
这种方式理论上可以管理无限多的后台程序。用户可以通过 `jobs, bg, fg` 等命令来查看/管理正在后台执行的程序。

在一些较为完备的shell中，键盘快捷键得到了很好的支持，用户可以通过 `Ctrl+Z` 来暂停/挂起正在执行的程序，并通过 `bg` 让其在后台恢复运行/ `fg` 让其恢复运行并提到前台。并且支持根据输入的程序暂停功能：在程序读取输入流时自动挂起。

运行结果

1. 复杂重定向和管道操作

```
 cat < dummy.mysh | wc > /dev/tty | echo "zy" > result.out | sha256sum | tr -d " -" >> result.out | wc | cat result.out | wc | cat result.out
```



2. 脚本执行

```
./MyShell.py -w dummy.mysh -a foo bar foobar hello world linux linus PyTorch CS231n
```

脚本内容 `dummy.mysh` ：

```
test ! -z ""
date +%s
time
set BOLD="\033[1m"
set RED="\033[31m"
set BLUE="\033[34m"
set RESET="\033[0m"

set LINE="$BOLD$BLUE##########################################################################$RESET"

echo $LINE

umask
echo "Changing UMASK to 0o777"
umask 777
umask
echo "Changing UMASK to 0o002"
umask 002

echo $LINE

echo "Hello, my name is $SHELL"
```

```
23  set hello_msg="Hello, my name is"
24  echo -r "$hello_msg $USER, and I live in $HOME"
25  echo "Should print sha256sum of zy in the next line"
26  echo "xz" | sha256sum | tr -d " -"
27  echo "Should print 1 1 65 in the following line"
28  echo "zy" | sha256sum | tr -d " -" | wc
29
30  echo $LINE
31
32  dir
33  pwd
34  unset hello_msg
35  echo "Should get empty output"
36  echo $hello_msg
37
38  echo $LINE
39
40  echo "Should print /dev/null"
41  ls /dev | grep null
42  echo "Should print all files containing 1 in /tmp"
43  ls /tmp | grep 1
44
45  echo $LINE
46
47  echo "Should make a file log.log"
48  echo "Hello, I'm your logger." > log.log
49  dir
50  echo "Should see content of log.log"
51  cat < log.log
52  echo "Hello, again ..." >> log.log
53  echo "Should display content of log.log"
54  cat log.log
55  echo "Should display word count of log.log"
56  wc < log.log
57
58  echo $LINE
59
60  echo "Opening some sleepy jobs"
61  echo "And calling command jobs"
62  sleep 2s | echo "Sleeping in $0" &
63  echo "waiting 0.25s"
64  jobs
65  sleep 0.25s
66  sleep 2s | echo "This is some job management" &
67  echo "waiting 0.25s"
68  jobs
69  sleep 0.25s
70  sleep 2s | echo "MyShell is $SHELL" &
71  echo "waiting 0.25s"
72  jobs
73  sleep 0.25s
74  sleep 2s &
75  echo "waiting 0.25s"
76  jobs
77  sleep 0.25s
78  sleep 2s &
79  echo "Getting current runnning jobs ..."
80  jobs
```

```
81
82    echo "Getting back to fore ground"
83    echo "Waiting for background jobs to terminate"
84    echo "At the same time I can still do other things like testing ... "
85    test "" -o "a"
86    test ! -z "a" -a ( -n "1" -o 1 -ge 1 ) -a 2 -ne 1
87    test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False, -a -o from
      right to left
88    test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # True
89    sleep 2s
90
91    echo "Should produce empty content"
92    jobs
93
94    echo "Jobs are done ~"
95
96    echo "Spawning dummy built_in job that is trying to read from user (will
      suspend)"
97
98    dummy &
99    dummy &
100   dummy &
101   dummy &
102
103   echo "Counting jobs"
104   jobs
105
106   echo "$RED$BOLD""WE'RE ONLY TERMINATING JOB [0] AND [1], YOU SHOULD SEE
      WARMING IF -w. NO ZOMBIE""$RESET"
107
108   term 0 1
109
110   echo $LINE
111
112   echo "calling environ ... "
113   environ
114
115   echo "Arg 0 is: $0"
116   echo "Arg 1 is: $1"
117   echo "Arg 2 is: $2"
118   echo "Arg 3 is: $3"
119   echo "Arg 4 is: $4"
120   echo "Arg 5 is: $5"
121   echo "Arg 6 is: $6"
122   echo "Arg 7 is: $7"
123   echo "Arg 8 is: $8"
124   echo "Arg 9 is: $9"
125
126   echo "Shifting number 1"
127   shift
128   echo "Arg 0 is: $0"
129   echo "Arg 1 is: $1"
130   echo "Arg 2 is: $2"
131   echo "Arg 3 is: $3"
132   echo "Arg 4 is: $4"
133   echo "Arg 5 is: $5"
134   echo "Arg 6 is: $6"
135   echo "Arg 7 is: $7"
```

```
    echo "Arg 8 is: $8"
    echo "Arg 9 is: $9"

    echo "Shifting number 2"
    shift
    echo "Arg 1 is: $1"

    echo "Shifting number 3"
    shift
    echo "Arg 1 is: $1"

    echo "Shifting number 4"
    shift
    echo "Arg 1 is: $1"

    echo "Shifting number 5"
    shift
    echo "Arg 1 is: $1"
    echo "$BOLD"Bye!"$RESET"
```

执行结果：

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:28:34 $ exit
 xuzh@ubuntu  ~/Projects/ShellDesign/MyShell  master ./MyShell.py -w dummy.mysh -a foo bar foob
ar hello world linux linus PyTorch CS231n
False
1596083355
2020-07-30 12:29:15.712186
############################################################
0o002
Changing UMASK to 0o777
0o777
Changing UMASK to 0o002
############################################################
Hello, my name is /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
Hello, my name is xuzh, and I live in /home/xuzh
Should print sha256sum of zy in the next line
b44f7d6b5283a44ee5f2bd98f84087a04810092122d75e8fbf8ad85f8f2981f1
Should print 1 1 65 in the following line
      1       1      65
############################################################
.
rwxrwxr-x 2020-07-30 12:08:45 hello
rw-rw-r-- 2020-07-29 14:28:40 log.log
rw-rw-r-- 2020-07-29 12:57:15 should.out
rw-rw-r-- 2020-07-29 12:36:49 MyShell.md
rw-rw-r-- 2020-07-30 12:26:17 result.out
rw-rw-r-- 2020-07-25 12:11:10 dummy.py
rwxrwxr-x 2020-07-25 14:09:29 process.py
rw-rw-r-- 2020-07-30 12:08:43 hello.c
rw-rw-r-- 2020-07-26 17:43:58 sleep10s.py
rwxrwxr-x 2020-07-29 14:52:17 MyShell.py
rw-rw-r-- 2020-07-29 12:56:43 dummy.mysh
rw-rw-r-- 2020-07-22 19:23:17 COLOR.py
rw-rw-r-- 2020-07-29 10:43:24 MyShellException.py
rwxrwxr-x 2020-07-29 12:42:10 __pycache__
~/Projects/ShellDesign/MyShell
Should get empty output
2020-07-30 12:29:15 ubuntu __main__[66739] WARNING Unable to get the varible "hello_msg", assigning e
mpty string

############################################################
```

…

```
    False
    1596083355
    2020-07-30 12:29:15.712186
    ############################################################
    0o002
    Changing UMASK to 0o777
```

```
 0o777
 Changing UMASK to 0o002
 ################################################################
 Hello, my name is /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
 Hello, my name is xuzh, and I live in /home/xuzh
 Should print sha256sum of zy in the next line
 b44f7d6b5283a44ee5f2bd98f84087a04810092122d75e8fbf8ad85f8f2981f1
 Should print 1 1 65 in the following line
       1       1      65
 ################################################################
 .
 rwxrwxr-x 2020-07-30 12:08:45 hello
 rw-rw-r-- 2020-07-29 14:28:40 log.log
 rw-rw-r-- 2020-07-29 12:57:15 should.out
 rw-rw-r-- 2020-07-29 12:36:49 MyShell.md
 rw-rw-r-- 2020-07-30 12:26:17 result.out
 rw-rw-r-- 2020-07-25 12:11:10 dummy.py
 rwxrwxr-x 2020-07-25 14:09:29 process.py
 rw-rw-r-- 2020-07-30 12:08:43 hello.c
 rw-rw-r-- 2020-07-26 17:43:58 sleep10s.py
 rwxrwxr-x 2020-07-29 14:52:17 MyShell.py
 rw-rw-r-- 2020-07-29 12:56:43 dummy.mysh
 rw-rw-r-- 2020-07-22 19:23:17 COLOR.py
 rw-rw-r-- 2020-07-29 10:43:24 MyShellException.py
 rwxrwxr-x 2020-07-29 12:42:10 __pycache__
 ~/Projects/ShellDesign/MyShell
 Should get empty output
 2020-07-30 12:29:15 ubuntu __main__[66739] WARNING Unable to get the
varible "hello_msg", assigning empty string

 ################################################################
 Should print /dev/null
 null
 Should print all files containing 1 in /tmp
 clr-debug-pipe-64261-7288941-in
 clr-debug-pipe-64261-7288941-out
 dotnet-diagnostic-64261-7288941-socket
 pymp-1yf6wit3
 pymp-i1l0r1ba
 ssh-3Dl1dZ2MVhMN
 ssh-5opVjzIOSY1n
 ssh-F4yJVhHqoOo1
 ssh-ssJdLj12gJhT
 systemd-private-32fa36ac417343a4813881b03c5a5a50-bolt.service-9WJsxv
 systemd-private-32fa36ac417343a4813881b03c5a5a50-colord.service-qxRv3v
 systemd-private-32fa36ac417343a4813881b03c5a5a50-fwupd.service-WNl3wS
 systemd-private-32fa36ac417343a4813881b03c5a5a50-ModemManager.service-
w89ayg
 systemd-private-32fa36ac417343a4813881b03c5a5a50-rtkit-daemon.service-
ebfv3E
 systemd-private-32fa36ac417343a4813881b03c5a5a50-systemd-
resolved.service-G0nqb0
 systemd-private-32fa36ac417343a4813881b03c5a5a50-systemd-
timesyncd.service-N7TNbO
 tmp-64015c9VBFgG4O3ca.tpl
 tmp-64015EunG2vwQ362u.tpl
 tmp-64015jJWFioLuj9YL.tpl
 vmware-root_1361-3988687315
```

```
vscode-ipc-0e8469e9-1fc8-467e-b42c-f6a8519ab561.sock
vscode-ipc-705a4712-c1f7-43b0-9465-99c771a42a1d.sock
vscode-ipc-76bfc346-b068-4fb1-8340-103432500cbb.sock
vscode-ipc-77ad59b8-138f-4875-884e-368ed7e31e7d.sock
vscode-ipc-846c3b8a-cd23-429d-aa84-b71bb5acbe7b.sock
vscode-ipc-a1c81be6-faa1-49b2-8a12-38a172ae37e9.sock
vscode-ipc-cb722aa7-90c1-4b82-b595-52d2e62b5d98.sock
vscode-ipc-e9b20dcd-2866-4d0b-bd69-426b2dc6b153.sock
vscode-ipc-efde013b-3ec4-4d77-9e0d-b10d25879dda.sock
vscode-typescript1000
####################################################################
Should make a file log.log
.
rwxrwxr-x 2020-07-30 12:08:45 hello
rw-rw-r-- 2020-07-30 12:29:15 log.log
rw-rw-r-- 2020-07-29 12:57:15 should.out
rw-rw-r-- 2020-07-29 12:36:49 MyShell.md
rw-rw-r-- 2020-07-30 12:26:17 result.out
rw-rw-r-- 2020-07-25 12:11:10 dummy.py
rwxrwxr-x 2020-07-25 14:09:29 process.py
rw-rw-r-- 2020-07-30 12:08:43 hello.c
rw-rw-r-- 2020-07-26 17:43:58 sleep10s.py
rwxrwxr-x 2020-07-29 14:52:17 MyShell.py
rw-rw-r-- 2020-07-29 12:56:43 dummy.mysh
rw-rw-r-- 2020-07-22 19:23:17 COLOR.py
rw-rw-r-- 2020-07-29 10:43:24 MyShellException.py
rwxrwxr-x 2020-07-29 12:42:10 __pycache__
Should see content of log.log
Hello, I'm your logger.
Should display content of log.log
Hello, I'm your logger.
Hello, again ...
Should display word count of log.log
      2       6      40
####################################################################
Opening some sleepy jobs
And calling command jobs
waiting 0.25s
[0] running env sleep 2s | echo "Sleeping in $0" &
waiting 0.25s
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
waiting 0.25s
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
[2] running env sleep 2s | echo "MyShell is $SHELL" &
waiting 0.25s
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
[2] running env sleep 2s | echo "MyShell is $SHELL" &
[3] running env sleep 2s &
Getting current runnning jobs ...
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
[2] running env sleep 2s | echo "MyShell is $SHELL" &
[3] running env sleep 2s &
[4] running env sleep 2s &
Getting back to fore ground
```

```
 Waiting for background jobs to terminate
 At the same time I can still do other things like testing ...
 True
 True
 False
 True
 Sleeping in /home/xuzh/Projects/ShellDesign/MyShell/dummy.mysh
 [0] finished env sleep 2s | echo "Sleeping in $0" &
 This is some job management
 [1] finished env sleep 2s | echo "This is some job management" &
 MyShell is /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
 [2] finished env sleep 2s | echo "MyShell is $SHELL" &
 [3] finished env sleep 2s &
 [4] finished env sleep 2s &
 Should produce empty content
 Jobs are done /home/xuzh
 Spawning dummy built_in job that is trying to read from user (will
suspend)
 builtin_dummy: before any input requirements
 [0] suspended env dummy &
 builtin_dummy: before any input requirements
 [1] suspended env dummy &
 builtin_dummy: before any input requirements
 [2] suspended env dummy &
 builtin_dummy: before any input requirements
 [3] suspended env dummy &
 Counting jobs
 [0] suspended env dummy &
 [1] suspended env dummy &
 [2] suspended env dummy &
 [3] suspended env dummy &
 WE'RE ONLY TERMINATING JOB [0] AND [1], YOU SHOULD SEE WARMING IF -w. NO
ZOMBIE
 [0] terminated env dummy &
 2020-07-30 12:29:18 ubuntu __main__[66819] WARNING Terminating job [0]
handler process by signal ...
 [1] terminated env dummy &
 2020-07-30 12:29:18 ubuntu __main__[66823] WARNING Terminating job [1]
handler process by signal ...
 ################################################################
 calling environ ...
 SSH_CONNECTION=192.168.28.1 12000 192.168.28.146 22
 LANG=en_US.UTF-8
 OLDPWD=/home/xuzh/Projects/ShellDesign/MyShell
 XDG_SESSION_ID=29
 USER=xuzh
 PWD=/home/xuzh/Projects/ShellDesign/MyShell
 HOME=/home/xuzh
 SSH_CLIENT=192.168.28.1 12000 22
 MAIL=/var/mail/xuzh
 SHELL=/home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
 SHLVL=2
 LOGNAME=xuzh
 DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
 XDG_RUNTIME_DIR=/run/user/1000
```

```
 PATH=/home/xuzh/.vscode-server/bin/91899dcef7b8110878ea59626991a18c8a6a1b3e/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/home/xuzh/.local/bin
 _=/home/xuzh/Projects/ShellDesign/MyShell/./MyShell.py
 VSCODE_IPC_HOOK_CLI=/tmp/vscode-ipc-a1c81be6-faa1-49b2-8a12-38a172ae37e9.sock
 TERM_PROGRAM=vscode
 TERM_PROGRAM_VERSION=1.47.3
 COLORTERM=truecolor
 VSCODE_GIT_IPC_HANDLE=/run/user/1000/vscode-git-2fbb053fa5.sock
 GIT_ASKPASS=/home/xuzh/.vscode-server/bin/91899dcef7b8110878ea59626991a18c8a6a1b3e/extensions/git/dist/askpass.sh
 VSCODE_GIT_ASKPASS_NODE=/home/xuzh/.vscode-server/bin/91899dcef7b8110878ea59626991a18c8a6a1b3e/node
 VSCODE_GIT_ASKPASS_MAIN=/home/xuzh/.vscode-server/bin/91899dcef7b8110878ea59626991a18c8a6a1b3e/extensions/git/dist/askpass-main.js
 TERM=xterm-256color
 ZSH=/home/xuzh/.oh-my-zsh
 PAGER=less
 LESS=-R
 LSCOLORS=Gxfxcxdxbxegedabagacad
 LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;31:*.tgz=01;31:*.arc=01;31:*.arj=01;31:*.taz=01;31:*.lha=01;31:*.lz4=01;31:*.lzh=01;31:*.lzma=01;31:*.tlz=01;31:*.txz=01;31:*.tzo=01;31:*.t7z=01;31:*.zip=01;31:*.z=01;31:*.Z=01;31:*.dz=01;31:*.gz=01;31:*.lrz=01;31:*.lz=01;31:*.lzo=01;31:*.xz=01;31:*.zst=01;31:*.tzst=01;31:*.bz2=01;31:*.bz=01;31:*.tbz=01;31:*.tbz2=01;31:*.tz=01;31:*.deb=01;31:*.rpm=01;31:*.jar=01;31:*.war=01;31:*.ear=01;31:*.sar=01;31:*.rar=01;31:*.alz=01;31:*.ace=01;31:*.zoo=01;31:*.cpio=01;31:*.7z=01;31:*.rz=01;31:*.cab=01;31:*.wim=01;31:*.swm=01;31:*.dwm=01;31:*.esd=01;31:*.jpg=01;35:*.jpeg=01;35:*.mjpg=01;35:*.mjpeg=01;35:*.gif=01;35:*.bmp=01;35:*.pbm=01;35:*.pgm=01;35:*.ppm=01;35:*.tga=01;35:*.xbm=01;35:*.xpm=01;35:*.tif=01;35:*.tiff=01;35:*.png=01;35:*.svg=01;35:*.svgz=01;35:*.mng=01;35:*.pcx=01;35:*.mov=01;35:*.mpg=01;35:*.mpeg=01;35:*.m2v=01;35:*.mkv=01;35:*.webm=01;35:*.ogm=01;35:*.mp4=01;35:*.m4v=01;35:*.mp4v=01;35:*.vob=01;35:*.qt=01;35:*.nuv=01;35:*.wmv=01;35:*.asf=01;35:*.rm=01;35:*.rmvb=01;35:*.flc=01;35:*.avi=01;35:*.fli=01;35:*.flv=01;35:*.gl=01;35:*.dl=01;35:*.xcf=01;35:*.xwd=01;35:*.yuv=01;35:*.cgm=01;35:*.emf=01;35:*.ogv=01;35:*.ogx=01;35:*.aac=00;36:*.au=00;36:*.flac=00;36:*.m4a=00;36:*.mid=00;36:*.midi=00;36:*.mka=00;36:*.mp3=00;36:*.mpc=00;36:*.ogg=00;36:*.ra=00;36:*.wav=00;36:*.oga=00;36:*.opus=00;36:*.spx=00;36:*.xspf=00;36:
 SSH_AUTH_SOCK=/tmp/ssh-sHY4pYyqIN0I/agent.66323
 SSH_AGENT_PID=66324
 PS1=$
 BOLD=\033[1m
 RED=\033[31m
 BLUE=\033[34m
 RESET=\033[0m
 LINE=\033[1m\033[34m##########################################################\033[0m
 Arg 0 is: /home/xuzh/Projects/ShellDesign/MyShell/dummy.mysh
 Arg 1 is: foo
 Arg 2 is: bar
 Arg 3 is: foobar
```

```
 Arg 4 is: hello
 Arg 5 is: world
 Arg 6 is: linux
 Arg 7 is: linus
 Arg 8 is: PyTorch
 Arg 9 is: CS231n
 Shifting number 1
 Arg 0 is: /home/xuzh/Projects/ShellDesign/MyShell/dummy.mysh
 Arg 1 is: bar
 Arg 2 is: foobar
 Arg 3 is: hello
 Arg 4 is: world
 Arg 5 is: linux
 Arg 6 is: linus
 Arg 7 is: PyTorch
 Arg 8 is: CS231n
 2020-07-30 12:29:19 ubuntu __main__[66739] WARNING Unable to get the varible "9", assigning empty string
 Arg 9 is:
 Shifting number 2
 Arg 1 is: foobar
 Shifting number 3
 Arg 1 is: hello
 Shifting number 4
 Arg 1 is: world
 Shifting number 5
 Arg 1 is: linux
 Bye!
 [2] terminated env dummy &
 [3] terminated env dummy &
 2020-07-30 12:29:19 ubuntu __main__[66827] WARNING Terminating job [2] handler process by signal ...
 2020-07-30 12:29:19 ubuntu __main__[66831] WARNING Terminating job [3] handler process by signal ...
```

3. 复杂 `test` 命令执行（同时检查注释功能）

```
 test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False, -a -o from right to left
 test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # True
```

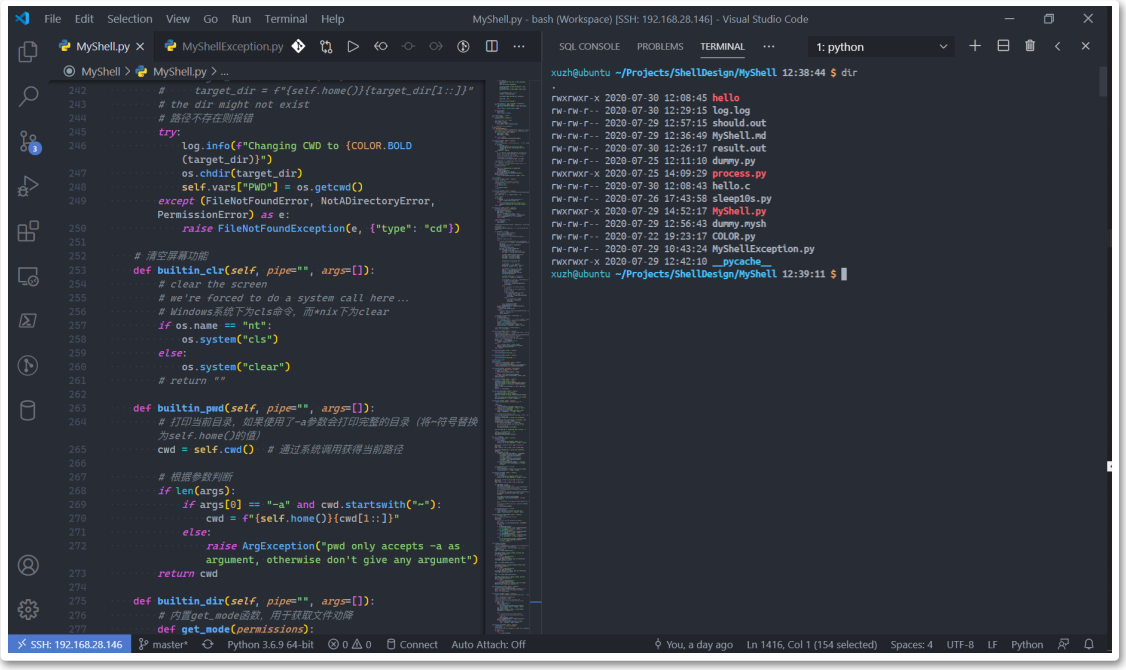
4. `cd`

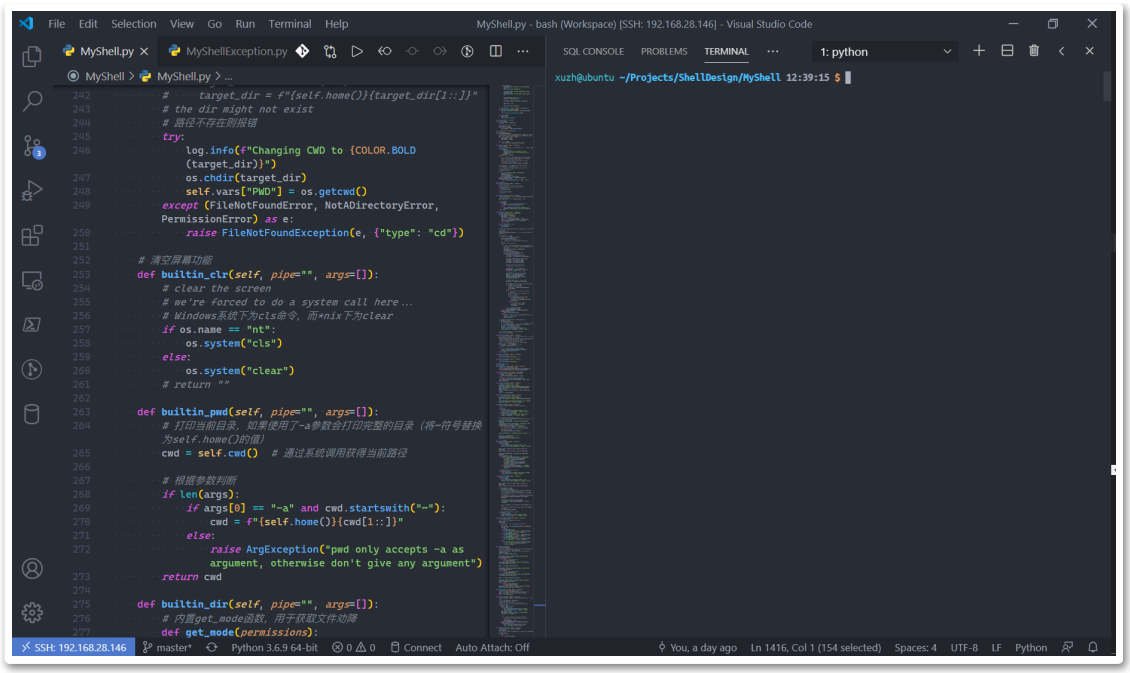
```
cd ..
cd ..
cd
cd DoesntExist
cd /dev
cd /var/log
cd .
cd ~
cd /
cd $HOME
```

```
xuzh@ubuntu ~/Projects/ShellDesign 12:37:02 $ cd ..
xuzh@ubuntu ~/Projects 12:37:07 $ cd ..
xuzh@ubuntu ~ 12:37:07 $ cd
~
xuzh@ubuntu ~ 12:37:07 $ cd DoesntExist
2020-07-30 12:37:07 ubuntu __main__[66882] ERROR Cannot successfully execute command "cd". Exception
is:
FileNotFoundException: [Errno 2] No such file or directory: 'DoesntExist'
Extra info: {'type': 'cd'}
xuzh@ubuntu ~ 12:37:07 $ cd /dev
xuzh@ubuntu /dev 12:37:07 $ cd /var/log
xuzh@ubuntu /var/log 12:37:07 $ cd .
xuzh@ubuntu /var/log 12:37:07 $ cd ~
xuzh@ubuntu ~ 12:37:07 $ cd /
xuzh@ubuntu / 12:37:07 $ cd $HOME
xuzh@ubuntu ~ 12:37:09 $
```

5. `clr`

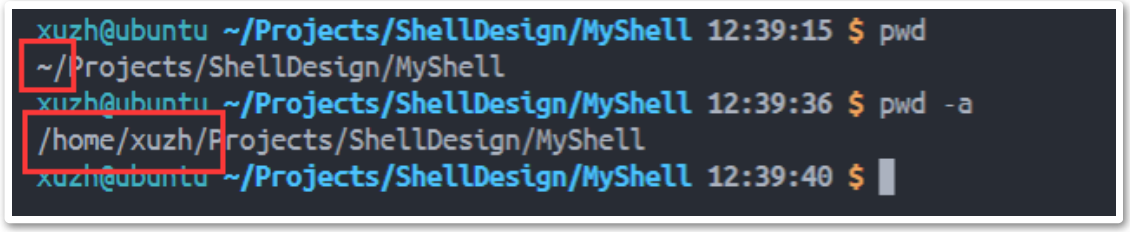
```
clr
```





6. `pwd`

```
pwd
pwd -a
```



7. `dir`

```
dir
cd ..
dir MyShell DirSync DoesntExist
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:39:40 $ dir
.
rwxrwxr-x 2020-07-30 12:08:45 hello
rw-rw-r-- 2020-07-30 12:29:15 log.log
rw-rw-r-- 2020-07-29 12:57:15 should.out
rw-rw-r-- 2020-07-29 12:36:49 MyShell.md
rw-rw-r-- 2020-07-30 12:26:17 result.out
rw-rw-r-- 2020-07-25 12:11:10 dummy.py
rwxrwxr-x 2020-07-25 14:09:29 process.py
rw-rw-r-- 2020-07-30 12:08:43 hello.c
rw-rw-r-- 2020-07-26 17:43:58 sleep10s.py
rwxrwxr-x 2020-07-29 14:52:17 MyShell.py
rw-rw-r-- 2020-07-29 12:56:43 dummy.mysh
rw-rw-r-- 2020-07-22 19:23:17 COLOR.py
rw-rw-r-- 2020-07-29 10:43:24 MyShellException.py
rwxrwxr-x 2020-07-29 12:42:10 __pycache__
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:40:37 $ cd ..
xuzh@ubuntu ~/Projects/ShellDesign 12:40:43 $ dir MyShell DirSync DoesntExist
MyShell
rwxrwxr-x 2020-07-30 12:08:45 hello
rw-rw-r-- 2020-07-30 12:29:15 log.log
rw-rw-r-- 2020-07-29 12:57:15 should.out
rw-rw-r-- 2020-07-29 12:36:49 MyShell.md
rw-rw-r-- 2020-07-30 12:26:17 result.out
rw-rw-r-- 2020-07-25 12:11:10 dummy.py
rwxrwxr-x 2020-07-25 14:09:29 process.py
rw-rw-r-- 2020-07-30 12:08:43 hello.c
rw-rw-r-- 2020-07-26 17:43:58 sleep10s.py
rwxrwxr-x 2020-07-29 14:52:17 MyShell.py
rw-rw-r-- 2020-07-29 12:56:43 dummy.mysh
rw-rw-r-- 2020-07-22 19:23:17 COLOR.py
rw-rw-r-- 2020-07-29 10:43:24 MyShellException.py
rwxrwxr-x 2020-07-29 12:42:10 __pycache__

DirSync
rwxrwxr-x 2020-07-22 19:23:17 DirSync.sh

DoesntExist
2020-07-30 12:40:59 ubuntu __main__[66882] ERROR Cannot successfully execute command "dir". Excepti
 is:
FileNotFoundException: Cannot list director[y|ies]:
[Errno 2] No such file or directory: 'DoesntExist'
Extra info: {'type': 'dir'}
xuzh@ubuntu ~/Projects/ShellDesign 12:40:59 $
```

8. `echo`

```
1    echo "\033[1m\033[31mHello, world.\033[0m"
2    echo "\033[1m\033[33mMy name is $SHELL\033[0m"
3    echo -r "\033[1m\031[31mMy name is $SHELL\033[0m"
4    echo without"$SHELL"any"$HOME"space and here come spaces
5    echo "中文测试" # 注释测试
```

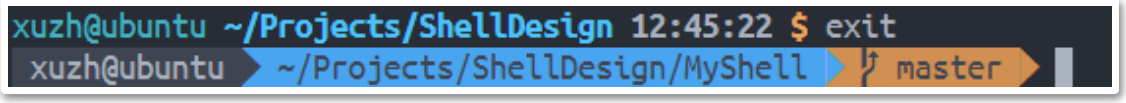
```
xuzh@ubuntu ~/Projects/ShellDesign 12:44:38 $ echo "\033[1m\033[31mHello, world.\033[0m"
Hello, world.
xuzh@ubuntu ~/Projects/ShellDesign 12:45:21 $ echo "\033[1m\033[33mMy name is $SHELL\033[0m"
My name is /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
xuzh@ubuntu ~/Projects/ShellDesign 12:45:21 $ echo -r "\033[1m\031[31mMy name is $SHELL\033[0m"
\033[1m\031[31mMy name is /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py\033[0m
xuzh@ubuntu ~/Projects/ShellDesign 12:45:21 $ echo without"$SHELL"any"$HOME"space and here come spaces
without/home/xuzh/Projects/ShellDesign/MyShell/MyShell.pyany/home/xuzhspace and here come spaces
xuzh@ubuntu ~/Projects/ShellDesign 12:45:22 $
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:55:59 $ echo "中文测试" # 注释测试
中文测试
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:08:03 $
```

9. `exit`

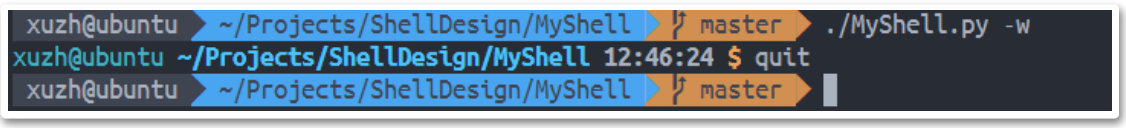
```
1    exit
```

```
xuzh@ubuntu ~/Projects/ShellDesign 12:45:22 $ exit
```



10. `quit`

```
quit
```



11. `jobs, fg, bg, term, exit` 任务管理

```
set BOLD="\033[1m"
set RED="\033[31m"
set BLUE="\033[34m"
set RESET="\033[0m"

set LINE="$BOLD$BLUE#################################################################$RESET"

echo $LINE

echo "Opening some sleepy jobs"
echo "And calling command jobs"
sleep 2s | echo "Sleeping in $0" &
echo "waiting 0.25s"
jobs
sleep 0.25s
sleep 2s | echo "This is some job management" &
echo "waiting 0.25s"
jobs
sleep 0.25s
sleep 2s | echo "MyShell is $SHELL" &
echo "waiting 0.25s"
jobs
sleep 0.25s
sleep 2s &
echo "waiting 0.25s"
jobs
sleep 0.25s
sleep 2s &
echo "Getting current runnning jobs ... "
jobs

echo "Getting back to fore ground"
echo "Waiting for background jobs to terminate"
echo "At the same time I can still do other things like testing ... "
test "" -o "a"
test ! -z "a" -a ( -n "1" -o 1 -ge 1 ) -a 2 -ne 1
test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False, -a -o from right to left
test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # True
sleep 2s

echo "Should produce empty content"
jobs

echo "Jobs are done ~"

```

```
46    echo "Spawning dummy built_in job that is trying to read from user (will
      suspend)"
47
48    dummy &
49    dummy &
50    dummy &
51    dummy &
52
53    echo "Counting jobs"
54    jobs
55
56    echo "$RED$BOLD""WE'RE ONLY TERMINATING JOB [0] AND [1], YOU SHOULD SEE
      WARMING IF -w. NO ZOMBIE""$RESET"
57
58    term 0 1
59
60    echo $LINE
61
62    exit # should see termination message
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:55 $ set BOLD="\033[1m"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ set RED="\033[31m"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ set BLUE="\033[34m"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ set RESET="\033[0m"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ set LINE="$BOLD$BLUE##################################
####################$RESET"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ echo $LINE
##################################################
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ echo "Opening some sleepy jobs"
Opening some sleepy jobs
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ echo "And calling command jobs"
And calling command jobs
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ sleep 2s | echo "Sleeping in $0" &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ echo "waiting 0.25s"
waiting 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ jobs
[0] running env sleep 2s | echo "Sleeping in $0" &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ sleep 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ sleep 2s | echo "This is some job management" &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ echo "waiting 0.25s"
waiting 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ jobs
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ sleep 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ sleep 2s | echo "MyShell is $SHELL" &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ echo "waiting 0.25s"
waiting 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ jobs
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
[2] running env sleep 2s | echo "MyShell is $SHELL" &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:56 $ sleep 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ sleep 2s &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ echo "waiting 0.25s"
waiting 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ jobs
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
[2] running env sleep 2s | echo "MyShell is $SHELL" &
[3] running env sleep 2s &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ sleep 0.25s
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ sleep 2s &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ echo "Getting current runnning jobs..."
Getting current runnning jobs...
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ jobs
[0] running env sleep 2s | echo "Sleeping in $0" &
[1] running env sleep 2s | echo "This is some job management" &
[2] running env sleep 2s | echo "MyShell is $SHELL" &
[3] running env sleep 2s &
[4] running env sleep 2s &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ echo "Getting back to fore ground"
Getting back to fore ground
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ echo "Waiting for background jobs to terminate"
Waiting for background jobs to terminate
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ echo "At the same time I can still do other things like te
sting..."
At the same time I can still do other things like testing...
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ test "" -o "a"
True
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ test ! -z "a" -a ( -n "1" -o 1 -ge 1 ) -a 2 -ne 1
True
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False,
-a -o from right to left
False
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # Tru
e
True
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:57 $ sleep 2s
Sleeping in /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
[0] finished env sleep 2s | echo "Sleeping in $0" &
This is some job management
[1] finished env sleep 2s | echo "This is some job management" &
MyShell is /home/xuzh/Projects/ShellDesign/MyShell/MyShell.py
[2] finished env sleep 2s | echo "MyShell is $SHELL" &
[3] finished env sleep 2s &
[4] finished env sleep 2s &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ echo "Should produce empty content"
Should produce empty content
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ jobs
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ echo "Jobs are done ~"
Jobs are done /home/xuzh
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ echo "Spawning dummy built_in job that is trying to read f
rom user (will suspend)"
Spawning dummy built_in job that is trying to read from user (will suspend)
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ dummy &
builtin_dummy: before any input requirements
[0] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ dummy &
builtin_dummy: before any input requirements
[1] suspended env dummy &
builtin_dummy: before any input requirements
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ dummy &
[2] suspended env dummy &
builtin_dummy: before any input requirements
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
[3] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ echo "Counting jobs"
Counting jobs
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ jobs
[0] suspended env dummy &
[1] suspended env dummy &
[2] suspended env dummy &
[3] suspended env dummy &
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ echo "$RED$BOLD""WE'RE ONLY TERMINATING JOB [0] ANDOMBIE""
$RESET"LD SEE WARMING IF -w. NO ZO
WE'RE ONLY TERMINATING JOB [0] AND [1], YOU SHOULD SEE WARMING IF -w. NO ZOMBIE
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ term 0 1
[0] terminated env dummy &
2020-07-30 12:48:59 ubuntu __main__[67251] WARNING Terminating job [0] handler process by signal...
[1] terminated env dummy &
2020-07-30 12:48:59 ubuntu __main__[67255] WARNING Terminating job [1] handler process by signal...
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ echo $LINE
##################################################
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:48:59 $ exit # should see termination message
[2] terminated env dummy &
2020-07-30 12:49:01 ubuntu __main__[67259] WARNING Terminating job [2] handler process by signal...
[3] terminated env dummy &
2020-07-30 12:49:01 ubuntu __main__[67263] WARNING Terminating job [3] handler process by signal...
 xuzh@ubuntu  ~/Projects/ShellDesign/MyShell  master
```

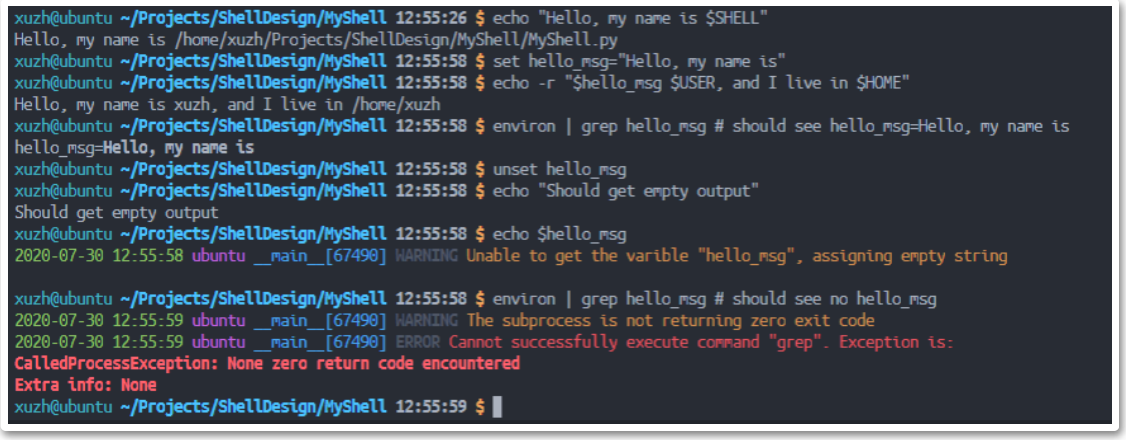
```
1    dummy &
2    dummy &
3    dummy &
4    dummy &
5    jobs
```

```
term 0 1
bg 2 3
term 2 3
jobs
dummy &
dummy &
fg 0
1
2
3
4
5
term 1
jobs
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:29 $ dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ dummy &
builtin_dummy: before any input requirements
[0] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ dummy &
builtin_dummy: before any input requirements
[1] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ dummy &
builtin_dummy: before any input requirements
[2] suspended env dummy &
builtin_dummy: before any input requirements
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ jobs
[3] suspended env dummy &
[0] suspended env dummy &
[1] suspended env dummy &
[2] suspended env dummy &
[3] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ term 0 1
[0] terminated env dummy &
2020-07-30 12:52:30 ubuntu __main__[67506] WARNING Terminating job [0] handler process by signal...
[1] terminated env dummy &
2020-07-30 12:52:30 ubuntu __main__[67510] WARNING Terminating job [1] handler process by signal...
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ bg 2 3
[2] continued env dummy &
[2] suspended env dummy &
[3] continued env dummy &
[3] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ term 2 3
[2] terminated env dummy &
2020-07-30 12:52:30 ubuntu __main__[67514] WARNING Terminating job [2] handler process by signal...
[3] terminated env dummy &
2020-07-30 12:52:30 ubuntu __main__[67518] WARNING Terminating job [3] handler process by signal...
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ jobs
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ dummy &
builtin_dummy: before any input requirements
[0] suspended env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ fg 0
builtin_dummy: before any input requirements
[0] continued env dummy &
[1] suspended env dummy &
[0] running env dummy &
dummy1> 1
1
dummy2> 2
dummy3> 3
dummyend> 4
dummy_content> 5
[0] finished env dummy &
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:30 $ term 1
[1] terminated env dummy &
2020-07-30 12:52:34 ubuntu __main__[67526] WARNING Terminating job [1] handler process by signal...
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:34 $ jobs
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 12:52:35 $
```

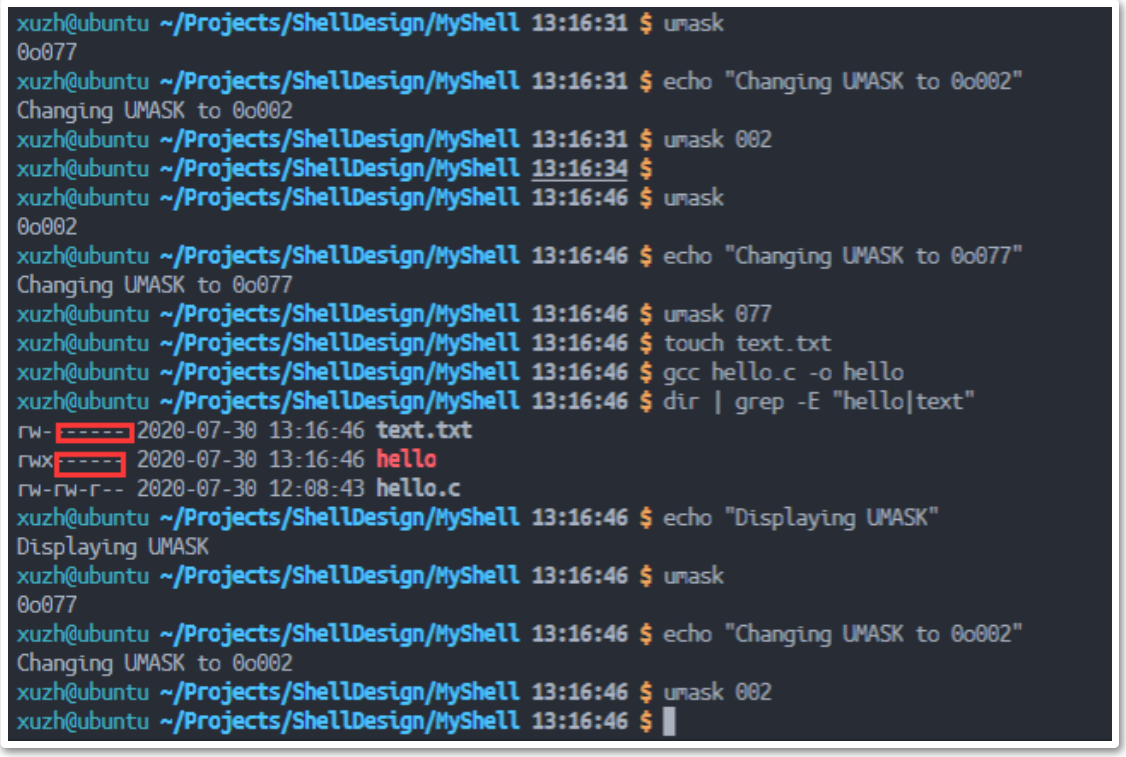
12. `environ, set, unset` 变量检查

```
echo "Hello, my name is $SHELL"
set hello_msg="Hello, my name is"
echo -r "$hello_msg $USER, and I live in $HOME"
environ | grep hello_msg # should see hello_msg=Hello, my name is
unset hello_msg
echo "Should get empty output"
echo $hello_msg
environ | grep hello_msg # should see no hello_msg
```



13. `umask`

```
umask
echo "Changing UMASK to 0o077"
umask 077
touch text.txt
gcc hello.c -o hello
dir | grep -E "hello|text"
echo "Displaying UMASK"
umask
echo "Changing UMASK to 0o002"
umask 002
```



14. `printio, exec`

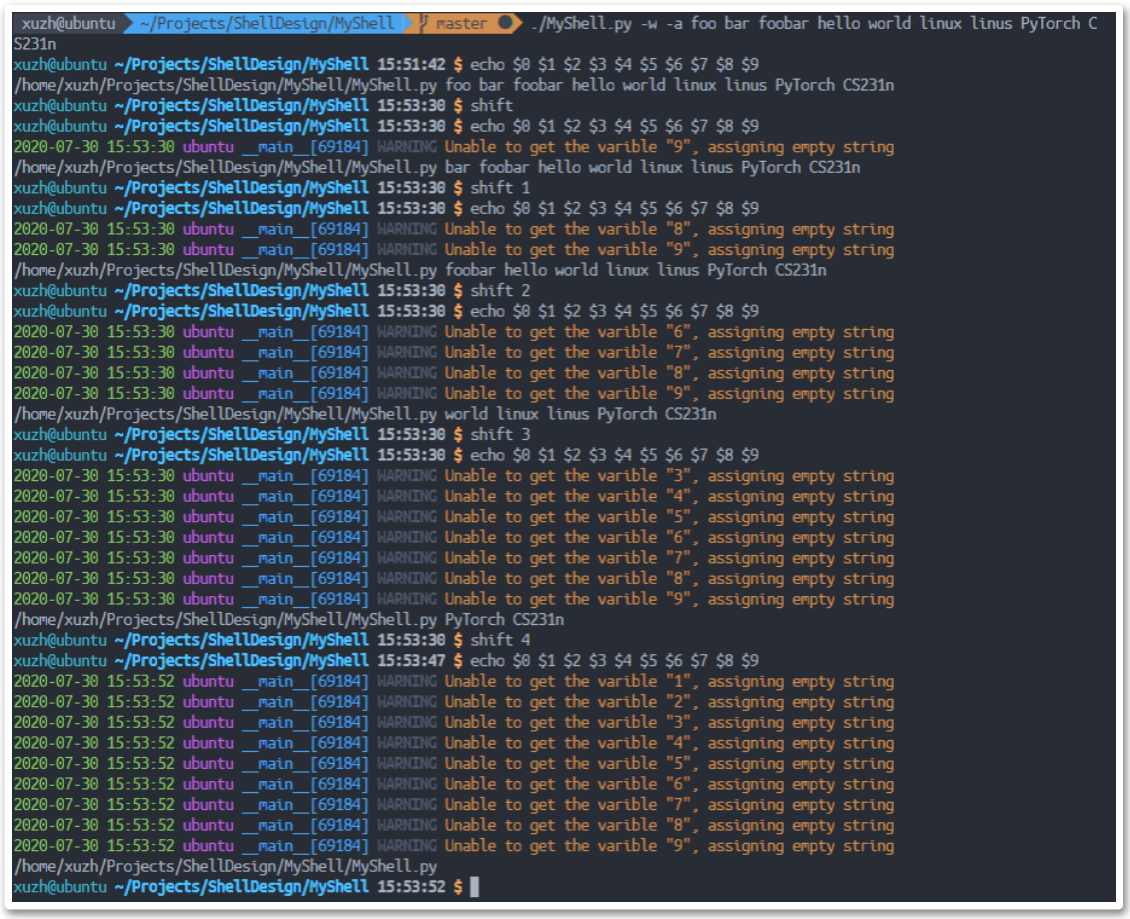
```
printio
exec < dummy.mysh > result.out
exec
printio
cat
exec < result.out > ""
printio
wc
exec < "" > ""
printio
exec > result.out
echo "REPLACING"
exec > ""
cat result.out
exec >> result.out
echo "APPENDING"
exec > ""
cat result.out
```

```
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:17 $ printio
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec < dummy.mysh > result.out
FILE NUMBER OF INPUT FILE: 4, redirecting MyShell input to <_io.TextIOWrapper name='dummy.mysh' mode='r' encoding='utf-8'>
FILE NUMBER OF OUTPUT FILE: 7, redirecting MyShell output to <_io.TextIOWrapper name='result.out' mode='w' encoding='utf-8'>
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec
FILE NUMBER OF INPUT FILE: 4, redirecting MyShell input to <_io.TextIOWrapper name='dummy.mysh' mode='r' encoding='utf-8'>
FILE NUMBER OF OUTPUT FILE: 7, redirecting MyShell output to <_io.TextIOWrapper name='result.out' mode='w' encoding='utf-8'>
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ printio
FILE NUMBER OF INPUT FILE: 4, redirecting MyShell input to <_io.TextIOWrapper name='dummy.mysh' mode='r' encoding='utf-8'>
FILE NUMBER OF OUTPUT FILE: 7, redirecting MyShell output to <_io.TextIOWrapper name='result.out' mode='w' encoding='utf-8'>
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ cat
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec < result.out > ""
FILE NUMBER OF INPUT FILE: 8, redirecting MyShell input to <_io.TextIOWrapper name='result.out' mode='r' encoding='utf-8'>
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ printio
FILE NUMBER OF INPUT FILE: 8, redirecting MyShell input to <_io.TextIOWrapper name='result.out' mode='r' encoding='utf-8'>
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ wc
 159  561 2940
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec < "" > ""
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ printio
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec > result.out
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 4, redirecting MyShell output to <_io.TextIOWrapper name='result.out' mode='w' encoding='utf-8'>
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ echo "REPLACING"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec > ""
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ cat result.out
REPLACING
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec >> result.out
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 4, redirecting MyShell output to <_io.TextIOWrapper name='result.out' mode='a' encoding='utf-8'>
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ echo "APPENDING"
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ exec > ""
FILE NUMBER OF INPUT FILE: 0, redirecting MyShell input to None
FILE NUMBER OF OUTPUT FILE: 1, redirecting MyShell output to None
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:18 $ cat result.out
REPLACING
APPENDING
xuzh@ubuntu ~/Projects/ShellDesign/MyShell 13:24:19 $
```

15. `shift`

通过命令行：`./MyShell.py -w dummy.mysh -a foo bar foobar hello world linux linus PyTorch CS231n` 运行MyShell

```
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
shift
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
shift 1
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
shift 2
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
shift 3
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
shift 4
echo $0 $1 $2 $3 $4 $5 $6 $7 $8 $9
```



16. `sleep`

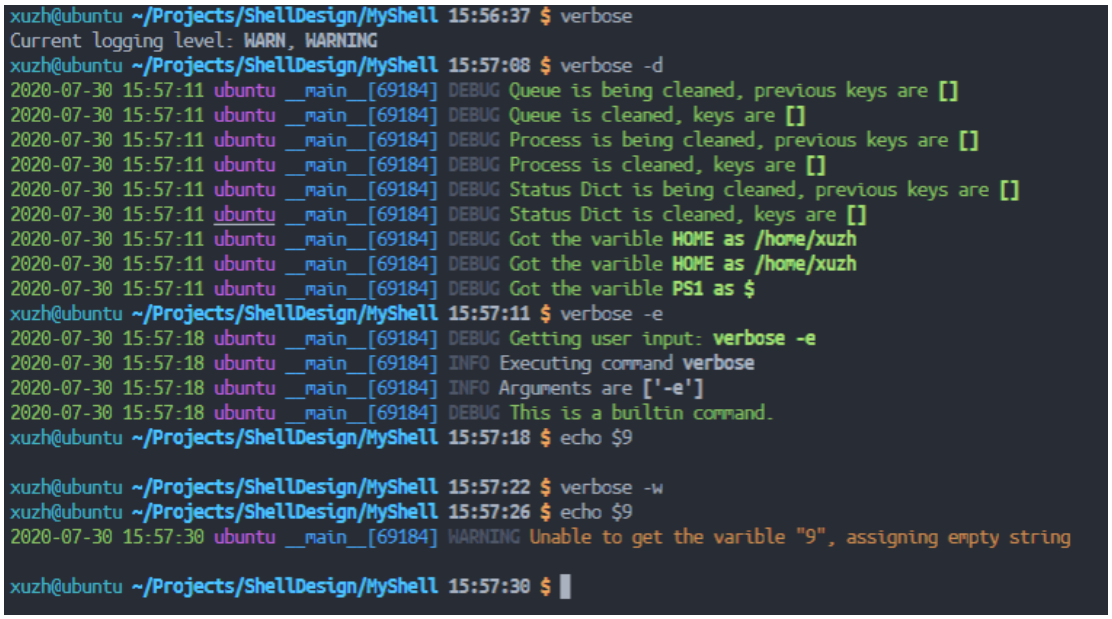
```
sleep 10s
```

17. `time`

```
time
```

18. `verbose`

```
verbose
verbose -d
verbose -e
echo $9
verbose -w
echo $9
```



19. `help`

```
help
help test
help MyShell
help jobs
help waitpid
help DoesntExist
```

SQL CONSOLE PROBLEMS OUTPUT TERMINAL ··· 1: python

shell 或者命令行解释器是操作系统中最基本的用户接口。我们实现了一个跨平台的简单的shell 程序—**MyShell**，它具有以下属性：

1. 支持的内部指令集：`cd, clr, pwd, dir, echo, exit, quit, jobs, fg, bg, term, environ, set, unset, umask, printio, exec, shift, test, sleep, time, help, verbose`

   请通过help command的格式查看指令的帮助

2. 开发者调试指令集：`dummy, check_zombie, queues`，用户一般不需要调用这些指令

   请通过help command的格式查看指令的帮助

3. MyShell在开始执行后会将环境变量`SHELL`设置为`MyShell`的运行位置

4. 其他的命令行输入被解释为程序调用，MyShell创建并执行这个程序，并作为自己的子进程。程序的执行的环境变量包含一下条目：

   `PARENT=<pathname>/MyShell.py`（也就是MyShell中`SHELL`变量的内容）。

5. MyShell能够从文件中提取命令行输入，例如shell 使用以下命令行被调用：

   ```shell
   ./MyShell.py dummy.mysh
   ```

   这个批处理文件应该包含一组命令集，当到达文件结尾时MyShell退出。很明显，如果MyShell被调用时没有使用参数，它会在屏幕上显示提示符请求用户输入。

6. MyShell除了上述的脚本执行，还支持其他运行时命令行参数：

   用户可以调用`./MyShell.py -h`查看相关内容

   ```shell
   usage: MyShell.py [-h] [-a [A [A ...]]] [-e] [-w] [-i] [-d] [F]

   MyShell by xudenden@gmail.com

   positional arguments:
     F              the batch file to be executed

   optional arguments:
     -h, --help     show this help message and exit
     -a [A [A ...]]  command line arguments to batch file
     -e             enable error level debugging info log
     -w             enable warning level debugging info log
     -i             enable info level debugging info log
     -d             enable debug(verbose) level debugging info log
   ```

   MyShell的调用举例：

   ```shell
   ./MyShell.py -w dummy.mysh -a foo bar foobar hello world linux linus PyTorch CS231n
   ```

7. MyShell支持I/O 重定向，stdin 和stdout，或者其中之一，例如命令行为：

   ```shell
   programname arg1 arg2 < inputfile > outputfile
   ```

   使用`arg1`和`arg2`执行程序`programname`，输入文件流被替换为`inputfile`，输出文件流被替换为`outputfile`。

   `stdout` 重定向持除了在上面注明需要打印信息（后台任务管理，输入输出重定向查看等）的所有内部指令。

   使用输出重定向时，如果重定向字符是`>`，则创建输出文件，如果存在则覆盖之；如果重定向字符为`>>`，也会创建输出文件，如果存在则添加到文件尾。

   对于`exec`指令，使用重定向符号会导致MyShell的输入输出被调整到指定的文件。

   MyShell在处理外部程序的调用时，为了方便用户观察结果和控制输入输出，会将`stderr`重定向到`stdout`一并打印到屏幕/定义的输出文件流。

8. MyShell支持后台程序执行。如果在命令行后添加`&`字符，在加载完程序后需要立刻返回命令行提示符。

   后台程序的主要管理接口为`jobs, term, fg, bg`

:

- `test`

  测试表达式结果是否为真或假

  `test expression`

  - 支持的表达式：

    `-o`：双目，逻辑或，参数为布尔值

    `-a`：双目，逻辑与，参数为布尔值

    `!`：单目，逻辑反，参数为布尔值

    `-z`：单目，字符串长度零检查，参数为字符串

    `-n`：单目，字符串长度非零检查，参数为字符串

    `==`：双目，字符串相等性检擦，参数为字符串

    `!=`：双目，字符出不等性检查，参数为字符串

    `-eq`：双目，数值相等性检查，参数为浮点数/整数

    `-ne`：双目，数值不等性检查，参数为浮点数/整数

    `-gt`：双目，数值大于性检查`lhs > rhs`，参数为浮点数/整数

    `-lt`：双目，数值小于性检查`lhs < rhs`，参数为浮点数/整数

    `-ge`：双目，数值大于等于检查`lhs >= rhs`，参数为浮点数/整数

    `-le`：双目，数值小于等于检查`lhs <= rhs`，参数为浮点数/整数

    `(`：左括号：被括号包裹的内容会被当成一个表达式来解释，返回布尔值

    `)`：右括号：被括号包裹的内容会被当成一个表达式来解释，返回布尔值

  - 表达式和运算子必须用空白符分隔开

  - 支持复杂的嵌套表达式，括号/单目运算符/双目运算符皆可嵌套执行

    ```shell
    test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False, -a -o from right to left
    test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # True
    ```

  - `-a, -o`从右向左结合，但用户可以通过括号来定制它们的运算顺序

  - 用户需要保证输入的内容是合理的可匹配的表达式

    - 括号需匹配完整
    - 运算符能处理的数据类型需要进行合理判断。所有经过运算的表达式结果：布尔值

(END)

附录

完整源码

```python
#!/usr/bin/python
import os  # 大部分os相关功能来自此包
import io  # 用于构建一个stdin的wrapper以捕捉程序的输入/输出请求
import sys  # 一些系统调用/常数的来源包
import re  # 正则表达式
import time  # 用在builtin_sleep中，针对Windows系统的睡眠操作
import platform  # 获取平台信息
import getpass  # 获取用户信息
import logging  # 用于打印一些详细调试信息
import coloredlogs  # 用于打印带颜色的调试信息
import datetime  # 用于取得时间/日期，用于builtin_time
import readline  # 用于提供人性化的input函数操作，例如输入历史功能等
import subprocess  # 用于刷出/控制子进程
import multiprocessing  # 用于后台程序的运行/管理
import traceback  # 用于打印报错信息的调用堆栈
```

```python
import argparse  # 用于解释本程序的命令行参数
import codecs  # 用于转义字符串，用于builtin_echo
import signal  # 用于在调用builtin_term后给multiprocessing传递信号
import copy  # 用于复制MyShell以进行multiprocessing和任务管理

from subprocess import Popen, PIPE, STDOUT  # 子进程管理
from multiprocessing import Process, Queue, Pipe, Pool, Manager, Value  # 多进程管理
from COLOR import COLOR  # 颜色信息
from MyShellException import *  # MyShell错误管理
log = logging.getLogger(__name__)

coloredlogs.install(level='DEBUG')  # Change this to DEBUG to see more info.


# a decorator for logging function call
# 装饰器：用于在函数调用之前打印相关信息，有助于调试或检查函数调用记录
def logger(func):
    def wrapper(*args, **kwargs):
        # 本装饰器中我们使用logging模块提供详细信息打印功能
        log.debug(f"CALLING FUNCTION: {COLOR.BOLD(func)}")
        return func(*args, **kwargs)
    return wrapper


# a class decorator that can modify all callable class method with a decorator
# here we use this to log function call of sys.stdin wrapper
# 一个带参数装饰器，用于对类进行修改：为类中的每一个可调用函数添加参数中所示的装饰器
def for_all_methods(decorator):
    def decorate(cls):
        for attr in cls.__dict__:  # there's propably a better way to do this
            if callable(getattr(cls, attr)):
                # 通过批量修改类的内容，避免重复代码，提高拓展性
                setattr(cls, attr, decorator(getattr(cls, attr)))
        return cls
    return decorate


# a dict that logs itself on every getting item operation
# 一个会在调用__get_item__前打印相关信息的函数
class LoggingDict(dict):
    def __getitem__(self, key):
        log.debug(f"GETTING DICT ITEM {COLOR.BOLD(key)}")
        return super().__getitem__(key)


class MyShell:
    def __init__(self, dict_in={}, cmd_args=[]):
        # 所有的内置功能的函数开头都是builtin_
        builtin_prefix = "builtin_"

        # builtins is a simple dict containing functions (not called on access)
        self.builtins = {}
        for key, value in MyShell.__dict__.items():
            if key.startswith(builtin_prefix):
                self.builtins[key[len(builtin_prefix)::]] = value

        # 环境变量，用到了下面将要提到的OnCallDict的功能
```

```
        # we're using callable dict to evaluate values on call
        unsupported = {str(i): MyShell.PicklableCMDArgs(self, i) for i in range(10)}
        self.vars = MyShell.OnCallDict(unsupported=unsupported)
        self.vars["PS1"] = "$"
        self.vars["SHELL"] = os.path.abspath(__file__)

        # 对于命令函参数，我们为了防止写太多重复代码，就使用了picklable_nested类来自动生成相关callable object
        # ! pickle cannot handle nested function, however a simulating callable object is fine
        # 由于Pickle无法处理nested function，我们使用类来实现相关功能
        if not cmd_args:
            cmd_args = [os.path.abspath(__file__)]
        self.cmd_args = cmd_args

        # update: 我们会在调用多线程/后台执行功能时清空这一部分的变量，能够解决的问题有：
        # 1. queue can only be pass by inheritance
        # 2. cannot pickle weakref object
        # 3. cannot pickle thread lock object
        # 4. for safety purpose, cannot pickle authentification string
        # pickle无法正常处理含有multiprocessing.Manager的object，因此我们不会显示储存这样的变量
        # ! damn strange ... when an object has a Manager, multiprocessing refuse to spawn it on Windows
        # self.job_manager = Manager()
        self.jobs = Manager().dict()
        self.status_dict = Manager().dict()
        self.queues = {}
        self.process = {}
        self.subp = None

        # 用于builtin_exec对输入输出的调整
        # None表示从原始输入/输出源接受相关信息
        self.input_file = None
        self.output_file = None

        # 初始化MyShell时可以传入初始环境变量
        for key, value in dict_in.items():
            # user should not be tampering with the vars already defined
            # however we decided not to disturb them
            self.vars[key] = value

        log.debug(f"SHELL var content: {self.vars['SHELL']}")
        log.debug(f"Built in command list: {self.builtins}")
        log.debug("MyShell is instanciated.")

    @property
    def job_counter(self):
        keys = [int(key) for key in self.jobs.keys()]
        count = 0
        while count in keys:
            count += 1
        return str(count)

    # 我们考虑过直接使用系统的环境变量接口，但那样或许就少了很多提前控制功能，跨平台性也不一定会很好
    # 所以MyShell会单独管理自己的变量（这里不一定要称为环境变量）
```

```
125         # 与MyShell配合使用，用于统一管理环境变量，并与真正的系统环境变量交互的字典
126         # 主要功能为，若字典value为可执行内容，则返回执行后的结果
127 
128         class OnCallDict():
129             def keys(self):
130                 return os.environ.keys()
131 
132             def __iter__(self):
133                 return os.environ.__iter__()
134 
135             def __init__(self, unsupported={}):
136                 self.unsupported = unsupported
137 
138             def __getitem__(self, key):
139                 try:
140                     if key in self.unsupported:
141                         var = self.unsupported[key]()
142                     else:
143                         var = os.environ.__getitem__(key)
144                     log.debug(f"Got the varible {COLOR.BOLD(key + ' as ' + var)}")
145                 except (KeyError, IndexError) as e:
146                     log.warning(f"Unable to get the varible \"{key}\", assigning
    empty string")
147                     var = ""
148                 return var
149 
150             def __setitem__(self, key, value):
151                 if key in self.unsupported:
152                     raise ReservedKeyException(f"Key {key} is reserved, along with
    \"{list(self.unsupported.keys())}\"")
153                 return os.environ.__setitem__(key, value)
154 
155             def __delitem__(self, key):
156                 if key in self.unsupported:
157                     raise ReservedKeyException(f"Key {key} is reserved, along with
    \"{list(self.unsupported.keys())}\"")
158                 return os.environ.__delitem__(key)
159 
160         # StdinWrapper是一个继承自io.TextWrapper，是本类型对获取输入的job进行线程暂停的一种
    方式
161         # 我们考虑过直接使用系统调用，但跨平台性会很难保证，因此对于后台外部命令，我们会直接关闭
    输入PIPE
162         @for_all_methods(logger)  # using the full class logging system
163         class StdinWrapper(io.TextIOWrapper):
164             def __init__(self, queue, count, job, status_dict, *args, **kwargs):
165                 super().__init__(*args, **kwargs)
166                 self.queue = queue
167                 self.job = job
168                 self.status_dict = status_dict
169                 self.count = count
170                 self.ok = False
171 
172             # 用于修改job状态并打印相关信息
173             def job_status(self, status):
174                 self.status_dict[self.count] = status
175                 print(f"{COLOR.BOLD(COLOR.YELLOW(f'[{self.count}]'))}
    {COLOR.BOLD(status)} env {COLOR.BOLD(self.job)}", file=sys.__stdout__)
176 
```

```
177             # 将第一次输入阻塞，等待主线程的fg命令
178             # there might exists a better way
179             def isok(self):
180                 if not self.ok:
181                     log.debug("Aha! You want to read something? Wait on!")
182                     self.job_status("suspended")
183 
184                     log.debug(f"WHAT IS SELF.STATUS_DICT: {self.status_dict}")
185                     log.debug(f"WHAT IS SELF.COUNT: {self.count}")
186 
187                     # queue的功能就是将本线程阻塞
188                     content = self.queue.get()
189 
190                     log.debug(f"WHAT DID YOU GET: {content}")
191 
192                     self.isok = True
193 
194                     self.job_status("running")
195 
196             # 为了方便调试，我们对很多内容的调用都内置了记录器
197             def __getattribute__(self, name):
198                 log.debug(f"GETTING ATTRIBUTE: {COLOR.BOLD(name)}")
199                 return super().__getattribute__(name)
200 
201             def fileno(self):
202                 self.isok()
203                 return super().fileno()
204 
205         # 以命令提示符方式执行MyShell
206         def __call__(self):
207             log.debug("This is MyShell.")
208 
209             # 通过返回值方式传输退出信号
210             exit_signal = False
211             while not exit_signal:
212                 exit_signal = self.command_prompt()
213 
214         # 用于模仿Nested Function的Object
215         # 可以被pickle
216         class PicklableCMDArgs():
217             # 在我们的使用环境下，shell变量被直接传入了self，也就保留了指针，因此外部对
    cmd_args变量的改变也会使得这里的结果不同
218             def __init__(self, shell, number):
219                 self.shell = shell
220                 self.number = number
221 
222             def __call__(self):
223                 return self.shell.cmd_args[self.number]
224 
225         def builtin_cd(self, pipe="", args=[]):
226             # 更换目录功能
227             # 对于多个参数的情况进行警告，并尝试进入第一个参数所示的功能
228             if len(args):
229                 if len(args) > 1:
230                     log.warning("Are you trying to cd into multiple dirs? We'll only
    accept the first argument.")
231                 target_dir = args[0]
232             else:
```

```
233                 # 在一般的shell中，无参数的cd调用会进入用户主目录
234                 # 但我们根据作业要求实现了打印当前工作目录的功能
235                 # well, we'd like to stay in home
236                 # but the teacher says we should pwd instead
237                 # target_dir = self.home()
238                 return self.builtin_pwd(pipe=pipe, args=args)
239 
240             # 若用户的输入开头为~符号，替换为用户主目录内容
241             # if target_dir.startswith("~"):
242             #     target_dir = f"{self.home()}{target_dir[1::]}"
243             # the dir might not exist
244             # 路径不存在则报错
245             try:
246                 log.info(f"Changing CWD to {COLOR.BOLD(target_dir)}")
247                 os.chdir(target_dir)
248                 self.vars["PWD"] = os.getcwd()
249             except (FileNotFoundError, NotADirectoryError, PermissionError) as e:
250                 raise FileNotFoundException(e, {"type": "cd"})
251 
252         # 清空屏幕功能
253         def builtin_clr(self, pipe="", args=[]):
254             # clear the screen
255             # we're forced to do a system call here...
256             # Windows系统下为cls命令，而*nix下为clear
257             if os.name == "nt":
258                 os.system("cls")
259             else:
260                 os.system("clear")
261             # return ""
262 
263         def builtin_pwd(self, pipe="", args=[]):
264             # 打印当前目录，如果使用了-a参数会打印完整的目录（将~符号替换为self.home()的值）
265             cwd = self.cwd()  # 通过系统调用获得当前路径
266 
267             # 根据参数判断
268             if len(args):
269                 if args[0] == "-a" and cwd.startswith("~"):
270                     cwd = f"{self.home()}{cwd[1::]}"
271                 else:
272                     raise ArgException("pwd only accepts -a as argument, otherwise
    don't give any argument")
273             return cwd
274 
275         def builtin_dir(self, pipe="", args=[]):
276             # 内置get_mode函数，用于获取文件劝降
277             def get_mode(permissions):
278                 map_string = "rwxrwxrwx"
279                 map_empty = "---------"
280                 # 我们使用列表生成器对bit列表进行判断
281                 result = "".join([map_string[i] if permissions[i] else map_empty[i]
    for i in range(9)])
282                 return result
283 
284             # 没有参数时，处理当前目录
285             if not len(args):
286                 args.append(".")
287 
288             # 不存在的目录的报错信息集合（用于最后的raise检查，会被调用'\n'.join）
```

```
289             not_in_list = []
290             # 储存将要打印的内容，最后会作为'\n'.join(result)的参数
291             result = []
292 
293             for target_dir in args:
294                 # 第一行是当前打印的目录的名称
295                 result.append(target_dir)
296                 # if target_dir.startswith("~"):
297                 #     target_dir = f"{self.home()}{target_dir[1::]}"
298                 # the dir might not exist
299                 try:
300                     # 可能会出现找不到文件的情况，将找不到的文件添加到not_in_list中，继续处理下一个目录的打印操作，并在最后raise
301                     log.debug(f"Listing dir {COLOR.BOLD(target_dir)}")
302                     # 通过os接口获得文件名称
303                     dir_list = os.listdir(target_dir)
304                     for file_name in dir_list:
305                         # 下面的stat等函数调用需要全路径
306                         full_name = f"{target_dir}/{file_name}"
307 
308                         # 文件的访问权限，最后修改时间等信息
309                         file_stat = os.stat(full_name)
310                         file_mode = file_stat.st_mode
311                         file_time = file_stat.st_mtime
312 
313                         # # guess we're not using this ...
314                         # type_code = file_mode >> 12
315 
316                         # 提取整数的每一位，一共获取9位
317                         permissions = [(file_mode >> bit) & 1 for bit in range(9 - 1, -1, -1)]
318 
319                         # 将访问权限和目录信息转换为字符串
320                         mode_str = get_mode(permissions)
321                         time_str = datetime.datetime.fromtimestamp(file_time).strftime("%Y-%m-%d %H:%M:%S")
322 
323                         # 合并成为最终结果的一行的前半部分
324                         file_line = f"{mode_str} {time_str} "
325 
326                         # 根据文件的类型/是否位可执行文件，写入相关颜色
327                         if os.path.isdir(full_name):
328                             # path is a directory
329                             file_line += COLOR.BLUE(COLOR.BOLD(file_name))
330                         else:
331                             # ! os.access not working properly on windows
332                             # todo: maybe we can recognize char block or others
333                             # this is brutal
334                             if os.access(full_name, os.X_OK):
335                                 # 用红色显示可执行文件
336                                 file_line += COLOR.RED(COLOR.BOLD(file_name))
337                             else:
338                                 # 用普通粗体显示一般文件
339                                 file_line += COLOR.BOLD(file_name)
340 
341                         # 添加到完整的result数组中
342                         result.append(file_line)
```

```
            except (FileNotFoundError, NotADirectoryError, PermissionError) as e:
                not_in_list.append(str(e))
            finally:
                # 若用户传入了多个参数，在不同的文件夹间打印一个换行（最终'\n'.join的效果）
                result.append("")

        # 找不到的目录（可能输入了一个文件）数量为非空
        if len(not_in_list):
            # 打印可以打印的内容
            # 由于raise会使函数停止继续执行，我们在此打印一些信息
            # ! 注意，这种情况下我们不会返回结果，piping会停止
            print("\n".join(result), end="")
            joined = '\n'.join(not_in_list)
            raise FileNotFoundException(f"Cannot list director[y|ies]:\n{joined}", {"type": "dir"})

        # 若一切正常则以字符串形式直接返回最终结果
        return "\n".join(result)

    def builtin_echo(self, pipe="", args=[]):
        # 我们支持-r参数，只能放在开头，如果用户使用了-r: raw input就不会对输入的特殊内容进行转义翻译
        # 但是相应的shell内置参数还是会被尝试替换，例如变量或者表示用户根目录的~符号，这不是echo函数所能控制的
        # 没有-r我们会尝试转义其他的escape code
        # ! if -r is provided, we won't do any escaping
        result = f"{' '.join(args)}\n"
        if len(args) >= 1 and args[0] == "-r":
            result = result[len("-r "):]
        else:
            # 通过codecs包实现较易于拓展的转义功能
            result = codecs.escape_decode(bytes(result, "utf-8"))[0].decode("utf-8")
        return result

    def builtin_exit(self, pipe="", args=[]):
        # 我们通过异常退出
        raise ExitException("Exitting ... ")

    def builtin_quit(self, pipe="", args=[]):
        # 我们通过异常退出
        raise ExitException("Quitting ... ")

    # 后台工作的相关内容
    @staticmethod
    def job_status_fmt(job_number, status, content):
        # 以统一的形式获得后台工作的样式
        return f"{COLOR.BOLD(COLOR.YELLOW(f'[{job_number}]'))} {COLOR.BOLD(status)} env {COLOR.BOLD(content)}"

    def job_status(self, job_number, status=None):
        # 设定后台工作的状态并返回统一的打印样式
        if status is not None:
            self.status_dict[job_number] = status
        else:
            status = self.status_dict[job_number]
```

```
394             return self.job_status_fmt(job_number, status, self.jobs[job_number])
395
396         def builtin_jobs(self, pipe="", args=[]):
397             # 打印当前所有后台工作的执行情况
398             log.debug("Trying to get all jobs")
399             log.info(f"Content of jobs {COLOR.BOLD(self.jobs)}")
400             log.info(f"Content of status_dict {COLOR.BOLD(self.status_dict)}")
401             result = [self.job_status(key) for key in self.jobs.keys()]
402             return '\n'.join(result)
403
404         def builtin_queues(self, pipe="", args=[]):
405             # 开发者测试用命令，检查内存泄漏用
406             log.debug("Trying to get all queues")
407             log.info(f"Content of queues {COLOR.BOLD(self.queues)}")
408             print(f"Existing multiprocessing.Queue s are:
    {COLOR.BOLD(self.queues)}", file=sys.__stdout__)
409
410         def builtin_fg(self, pipe="", args=[]):
411             # 将因为尝试获取用户输入而挂起的工作提到前台执行
412
413             # 数量检查
414             if len(args) != 1:
415                 raise JobException("Argument number is not correct, only one
    expected.", {"type": "len"})
416             elif args[0] not in self.jobs:
417                 raise JobException(f"Cannot find job is jobs number \"{args[0]}\".",
    {"type": "key"})
418
419             # 通过multiprocessin.Queue进行沟通
420             # ! 注意，外部命令的读取请求不会被处理，因为在后台执行的subshell中PIPE会被关闭
421             log.debug(f"Foreground is called with {COLOR.BOLD(args)}")
422             print(self.job_status_fmt(args[0], "continued", self.jobs[args[0]]),
    file=sys.__stdout__)
423             if self.status_dict[args[0]] == "suspended":
424                 # ! only put into queue if already suspended
425                 # else the main process will get what it just put into the queue
426                 self.queues[args[0]].put("dummy")
427
428             log.info("Waiting for foreground task to finish ... ")
429
430             # 挂起主线程，等待后台程序执行完毕
431             self.queues[args[0]].get()
432             log.debug("Continuing main process")
433
434         # 继续执行后台job
435         def builtin_bg(self, pipe="", args=[]):
436             # 处理相关参数调用
437             if not len(args):
438                 raise JobException("Argument number is not correct, one or more
    expected.", {"type": "len"})
439
440             not_in_list = [arg for arg in args if arg not in self.jobs]
441             args = [arg for arg in args if arg in self.jobs]
442
443             log.debug(f"Background is called with {COLOR.BOLD(args)}")
444             for job_number in args:
445                 if self.status_dict[job_number] == "suspended":
```

```
446                 print(self.job_status_fmt(job_number, "continued",
    self.jobs[job_number]), file=sys.__stdout__)
447                 print(self.job_status_fmt(job_number, "suspended",
    self.jobs[job_number]), file=sys.__stdout__)
448             elif self.status_dict[job_number] == "running":
449                 log.warning(f"Job [{job_number}] is already running")
450 
451         # 找不到的后台工作会被反馈给用户
452         if len(not_in_list):
453             raise JobException(f"Cannot find job with number \"
    {not_in_list}\".", {"type": "key"})
454 
455     def builtin_term(self, pipe="", args=[]):
456         # 处理相关参数调用
457         if not len(args):
458             raise JobException("Argument number is not correct, one or more
    expected.", {"type": "len"})
459 
460         not_in_list = [arg for arg in args if arg not in self.jobs]
461         args = [arg for arg in args if arg in self.jobs]
462 
463         for job_number in args:
464             log.debug("Terminating ... ")
465             # 对multiprocessing的进程包发出signal.SIGTERM信号，给其机会处理相关内容
    （关闭subshell等）
466             # ! 在*nix上与subshell，run_command_wrap配合可避免zombie process
467             # ! Windows中由于接口不匹配的原因，无法完全清除zombie
468             os.kill(self.process[job_number].pid, signal.SIGTERM)
469 
470             print(self.job_status_fmt(job_number, "terminated",
    self.jobs[job_number]), file=sys.__stdout__)
471 
472             # 我们以jobs数组为蓝本，判断jobs是否已完成或者被强行结束等
473             # 配合clean_up函数，status_dict和queues等其他数组会被正常清理
474             del self.jobs[job_number]
475 
476         # 找不到的后台工作会被反馈给用户
477         if len(not_in_list):
478             raise JobException(f"Cannot find jobs with number \"
    {not_in_list}\".", {"type": "key"})
479 
480     def builtin_check_zombie(self, pipe="", args=[]):
481         # 开发者调试用函数
482         # 用于检查当前进程下的zombie process
483         any_process = -1
484         while True:
485             # This will raise an exception on Windows.  That's ok.
486             pid, status = os.waitpid(any_process, os.WNOHANG)
487             if pid == 0:
488                 break
489             if os.WIFEXITED(status):
490                 print(f"The process of pid \"{COLOR.BOLD(pid)}\" is exited.",
    file=sys.__stdout__)
491             elif os.WIFSTOPPED(status):
492                 print(f"The process of pid \"{COLOR.BOLD(pid)}\" is stopped.",
    file=sys.__stdout__)
493             elif os.WIFCONTINUED(status):
```

```
494                 print(f"The process of pid \"{COLOR.BOLD(pid)}\" is continued.",
    file=sys.__stdout__)
495             else:
496                 print(f"The process of pid \"{COLOR.BOLD(pid)}\" is of status
    {COLOR.BOLD(status)}", file=sys.__stdout__)
497
498     def cleanup_jobs(self):
499         # 对jobs命令的拓展
500         # 我们的任务管理系统以self.jobs字典为准，每次命令调用后都会刷新当前的数组内容以删除
    不必要的元素
501         # 管理：self.queues, self.status_dict, self.process
502         # 避免了内存泄漏
503         keys = list(self.queues.keys())
504
505         log.debug(f"Queue is being cleaned, previous keys are
    {COLOR.BOLD(keys)}")
506         for i in keys:
507             if i not in self.jobs.keys():
508                 del self.queues[i]
509         log.debug(f"Queue is cleaned, keys are
    {COLOR.BOLD(list(self.queues.keys()))}")
510
511         keys = list(self.process.keys())
512
513         log.debug(f"Process is being cleaned, previous keys are
    {COLOR.BOLD(keys)}")
514         for i in keys:
515             if i not in self.jobs.keys():
516                 del self.process[i]
517         log.debug(f"Process is cleaned, keys are
    {COLOR.BOLD(list(self.process.keys()))}")
518
519         keys = list(self.status_dict.keys())
520
521         log.debug(f"Status Dict is being cleaned, previous keys are
    {COLOR.BOLD(keys)}")
522         for i in keys:
523             if i not in self.jobs.keys():
524                 del self.status_dict[i]
525         log.debug(f"Status Dict is cleaned, keys are
    {COLOR.BOLD(list(self.status_dict.keys()))}")
526
527     def builtin_environ(self, pipe="", args=[]):
528         # # 将MyShell的环境变量返回给用户
529         result = [f"{key}={COLOR.BOLD(self.vars[key])}" for key in self.vars] +
    [""]  # for dummy line break
530         return "\n".join(result)
531
532     def builtin_set(self, pipe="", args=[]):
533         # 修改脚本变量，用户需要保证自己的输入内容被解释为一个完整的参数
534         # 用等于号分割变量名和变量的具体值
535         # 变量都以字符串的形式储存
536         # we'd like the user to use the equal sign
537         # and we'd like to treat varibles only as string
538         for arg in args:
539             split = arg.split("=")
540             if len(split) != 2:
```

```
                raise SetPairUnmatchedException(f"Cannot match argument {arg}.", {"type": "set"})
            key, value = split
            log.debug(f"Setting \"{key}\" in environment variables to \"{value}\"")

            try:
                # 修改pwd时，可能会有找不到目录的错误
                # might be error since self.vars is not a simple dict
                self.vars[key] = value
            except (FileNotFoundError, NotADirectoryError, PermissionError) as e:
                raise FileNotFoundException(e, {"type": "set", "arg_pair": [key, value]})

    def builtin_unset(self, pipe="", args=[]):
        # 取消MyShell中一些已经设置好的变量
        cannot_unset = []
        # 类似builtin_environ中的处理方式，为了跳过可能出错的变量，我们手动循环
        for key in args:
            try:
                del self.vars[key]
            except (KeyError, ReservedKeyException) as e:
                cannot_unset.append(key)
        if len(cannot_unset):
            raise UnsetKeyException(f"Cannot find/unset these keys: {cannot_unset}", {"keys": cannot_unset})

    def builtin_umask(self, pipe="", args=[]):
        # 设置系统umask的值
        # subshell中的命令也会继续采用这一umask
        # 若没有任何参数，则直接打印umask
        # ! 在Windows上无法正常修改umask
        if len(args) > 1:
            # 参数数量为0或者1
            raise UmaskException("Argument number is not correct, only one or zero expected.", {"type": "len"})
        elif len(args) == 1:
            try:
                # 用户输入的umask值不一定有效
                log.debug(f"Value of ARG: {COLOR.BOLD(args[0])}")
                umask = int(args[0], 8)
                log.debug(f"Value of ARG in int of 8: {COLOR.BOLD(umask)}")
                os.umask(umask)
            except ValueError as e:
                raise UmaskException(e, {"type": "value"})
        else:
            # dummy parameter
            old = os.umask(0)
            log.debug(f"Value of OLD: {COLOR.BOLD(old)}")
            os.umask(old)
            # 返回umask的时候采用补零3位8进制格式
            return "0o{:03o}".format(old)

    def builtin_printio(self, pipe="", args=[]):
        # 开发者调试用命令，用于检查当前的exec文件重定向
        # note: 由于我们要检查重定向，这里会直接打印到stdout而非输入输出文件
        result = []
```

```
593             result.append(f"FILE NUMBER OF INPUT FILE:
    {COLOR.BOLD(self.input_file.fileno() if self.input_file is not None else
    sys.__stdin__.fileno())}, redirecting MyShell input to
    {COLOR.BOLD(self.input_file)}")
594             result.append(f"FILE NUMBER OF OUTPUT FILE:
    {COLOR.BOLD(self.output_file.fileno() if self.output_file is not None else
    sys.__stdout__.fileno())}, redirecting MyShell output to
    {COLOR.BOLD(self.output_file)}")
595             # ! debugging command, no redirection
596             print("\n".join(result), file=sys.__stdout__)
597
598         def builtin_exec(self, pipe="", args=[]):
599             # exec命令会修改MyShell命令的输入输出源
600             # note: 由于此命令的特殊性，我们会在该函数得到执行的上一层加入一些逻辑
601             if len(args) != 3:
602                 log.critical("Internal error, builtin_exec should be executed with
    exactly three arguments")
603                 raise ArgException("Internal error, builtin_exec should be executed
    with exactly three arguments")
604
605             log.debug(f"FILE NUMBER OF SYS.__STDIN__:
    {COLOR.BOLD(sys.__stdin__.fileno())}")
606             log.debug(f"FILE NUMBER OF SYS.__STDOUT__:
    {COLOR.BOLD(sys.__stdout__.fileno())}")
607             log.debug(f"FILE NUMBER OF SYS.__STDERR__:
    {COLOR.BOLD(sys.__stderr__.fileno())}")
608
609             # 我们刚刚已经保证走到这一步的参数数量为2
610             if args[0]:  # 若非空，尝试设置输入流
611                 try:
612                     # the function open will automatically raise FileNotFoundError
613                     new_file = open(args[0], "r", encoding="utf-8")
614
615                     # 新文件打开没有出错时我们才会关闭/修改旧文件
616                     # 无论如何，关闭原来已经打开的输入输出文件
617                     if self.input_file is not None:
618                         self.input_file.close()
619                     self.input_file = new_file
620                     log.debug(f"FILE NUMBER OF INPUT FILE:
    {COLOR.BOLD(self.input_file.fileno())}")
621                 except FileNotFoundError as e:
622                     raise FileNotFoundException(e, {"type": "redi_in"})
623             elif args[0] is None:
624                 log.debug("Doing nothing...")
625             else:
626                 log.debug(f"Setting input file to None: stdin")
627                 self.input_file = None
628
629             if args[1]:  # 若非空，尝试设置输入流
630                 try:
631                     # the function open will automatically raise FileNotFoundError
632                     new_file = open(args[1],  "a" if args[2] else "w",
    encoding="utf-8")
633
634                     # 新文件打开没有出错时我们才会关闭/修改旧文件
635                     if self.output_file is not None:
636                         self.output_file.close()
637                     self.output_file = new_file
```

```
638                 log.debug(f"FILE NUMBER OF OUTPUT FILE:
    {COLOR.BOLD(self.output_file.fileno())}")
639             except FileNotFoundError as e:
640                 raise FileNotFoundException(e, {"type": "redi_out",
    "redi_append": args[2]})
641         elif args[1] is None:
642             log.debug("Doing nothing ... ")
643         else:
644             log.debug(f"Setting output file to None: stdout")
645             self.output_file = None
646 
647         self.builtin_printio()
648 
649     def builtin_shift(self, pipe="", args=[]):
650         # 内置shift功能，在脚本调用时尤其有用
651         log.debug(f"Shift args are {COLOR.BOLD(args)}")
652         log.debug(f"Previously cmd_args are {COLOR.BOLD(self.cmd_args)}")
653 
654         if len(args) > 1:
655             raise ArgException(f"Expecting zero or one argument(s), got
    {len(args)}", {"args": args})
656         elif not args:
657             # 空参数情况移动1位
658             args.append(1)
659 
660         # $0会被保留
661         dollar_zero = self.cmd_args[0]
662 
663         try:
664             # 此时的shift时包含dollar_zero的
665             int_val = int(args[0])
666             if int_val < 0:
667                 raise ValueError("Non-negative int value expected")
668             self.cmd_args = self.cmd_args[int_val::]
669         except IndexError:
670             pass
671         except ValueError as e:
672             raise ArgException("Non-negative int convertible argument expected.
    {e}", {"args": args})
673 
674         # 替换掉dollar_zero位置的元素
675         if not len(self.cmd_args):
676             self.cmd_args = [dollar_zero]
677         else:
678             self.cmd_args[0] = dollar_zero
679 
680         log.debug(f"Now cmd_args are {COLOR.BOLD(self.cmd_args)}")
681 
682     def builtin_test(self, pipe="", args=[]):
683         # ! test函数的结合方向是从右向左
684         # todo: 实现带运算符顺序处理的逆波兰表达式（现在对-a和-o是一视同仁的）
685         # 结合递归调用
686         # builtin_test功能中使用到的等级函数
687         # 主要用于对操作符进行分类，方便调用，例如1，3为单目运算符
688         level = {
689             "(": 0, ")": 0,
690             "-z": 1, "-n": 1,
```

```
691                 "=": 2, "≠": 2, "-eq": 2, "-ge": 2, "-gt": 2, "-le": 2, "-lt": 2,
    "-ne": 2,
692                 "!": 3,
693                 "-a": 4, "-o": 4,
694             }
695
696             # 单目运算符操作
697             def test_unary(operator, operand):
698                 if operator == "-z":
699                     return not len(str(operand))
700                 if operator == "-n":
701                     return len(str(operand))
702                 if operator == "!":
703                     return not bool(operand)
704
705                 log.critical("Unrecoginized operator in a place it shouldn't be")
706                 raise TestException(f"Unrecognized unary operator \"{operator}\"")
707
708             # 双目运算符操作
709             def test_binary(op, lhs, rhs):
710                 if op == "=":
711                     return str(lhs) == str(rhs)
712                 if op == "≠":
713                     return str(lhs) != str(rhs)
714                 if op == "-eq":
715                     # todo: exception
716                     return float(lhs) == float(rhs)
717                 if op == "-ge":
718                     return float(lhs) >= float(rhs)
719                 if op == "-gt":
720                     return float(lhs) > float(rhs)
721                 if op == "-le":
722                     return float(lhs) <= float(rhs)
723                 if op == "-lt":
724                     return float(lhs) < float(rhs)
725                 if op == "-ne":
726                     return float(lhs) != float(rhs)
727                 if op == "-a":
728                     return bool(lhs) and bool(rhs)
729                 elif op == "-o":
730                     return bool(lhs) or bool(rhs)
731
732                 log.critical("Unrecoginized operator in a place it shouldn't be")
733                 raise TestException(f"Unrecognized binary operator \"{operator}\"")
734
735             def expand_expr(args):
736                 # ! we combine from the right, that is the right most value are
    evaluated first
737                 try:
738                     ind = 0
739                     if len(args) == 1:
740                         return args[0]
741
742                     # 非运算符，视为双目运算
743                     if args[ind] not in level:
744                         lhs = args[ind]
745                         op = args[ind+1]
746                         # 这句话保证了从右向左结合
```

```
                    rhs = expand_expr(args[ind+2::])
                    return test_binary(op, lhs, rhs)

                # match parentheses
                if args[ind] == "(":
                    # 对于小括号，我们将其视为一个整体
                    org = ind
                    count = 1
                    while count:
                        ind += 1
                        if args[ind] == "(":
                            count += 1
                        elif args[ind] == ")":
                            count -= 1
                    lhs = expand_expr(args[org+1:ind])
                    if ind == len(args)-1:
                        return lhs
                    op = args[ind+1]
                    # 这句话保证了从右向左结合
                    rhs = expand_expr(args[ind+2::])
                    return test_binary(op, lhs, rhs)

                if level[args[ind]] in [1, 3]:
                    # 对于单目运算符，我们也将其视为一个整体
                    op = args[ind]
                    oa, ind = get_one(args[ind+1::])
                    lhs = test_unary(op, oa)
                    if ind == len(args)-1:
                        return lhs
                    op = args[ind+1]
                    # 这句话保证了从右向左结合
                    rhs = expand_expr(args[ind+2::])
                    return test_binary(op, lhs, rhs)

            # note: 在此处理整个函数调用过程中可能的错误
            # int，bool无法转换/解释等
            except (ValueError, KeyError) as e:
                # log.error(f"{e}")
                raise TestException(e)

        def get_one(args):
            # 获取一位bool值，并返回下一个有效值的位置
            ind = 0
            if len(args) == 1 or args[ind] not in level:
                return args[0], 1

            if args[ind] == "(":
                org = ind
                while args[ind] != ")":
                    ind += 1
                return expand_expr(args[org+1:ind]), ind+1

            if level[args[ind]] in [1, 3]:
                op = args[ind]
                oa, ind = get_one(args[ind+1::])
                return test_unary(op, oa), ind+1

        # we can only use string to pass values
```

```
805             # 返回布尔值的字符串表示
806             try:
807                 result = str(bool(expand_expr(args)))
808                 return result
809             except IndexError as e:
810                 raise TestException(f"Unrecognized test expression, check your
    syntax. {e}")
811 
812         def builtin_sleep(self, pipe="", args=[]):
813             if os.name == "nt":
814                 # ! Windows系统没有相应的sleep命令，但Python存在相应的接口
815                 try:
816                     if len(args) != 1:
817                         raise ArgException("Exactly one argument expeceted")
818                     if args[0].endswith("s"):
819                         value = float(args[0][0:-1])
820                     else:
821                         value = float(args[0])
822                     time.sleep(value)
823                 except ValueError as e:
824                     raise SleepException(f"Unrecoginized sleep time format, are you
    on NT? Use second as unit and put s at the back of the time string. {e}")
825             else:
826                 self.subshell(target="sleep", args=args, pipe=pipe)
827 
828         def builtin_time(self, pipe="", args=[]):
829             # 显示当前的时间
830             return str(datetime.datetime.now())
831 
832         def builtin_dummy(self, pipe="", args=[]):
833             # 一个内置的dummy命令，用于测试是否可以正常触发suspension
834             print("builtin_dummy: before any input requirements")
835             print(input("dummy1> "))
836             print(input("dummy2> "))
837             print(input("dummy3> "))
838             print(input("dummyend> "))
839             result = input("dummy_content> ")
840             return result
841 
842         def builtin_help(self, pipe="", args=[]):
843             # 在线帮助函数
844             # todo: 写好在线帮助
845             help_dict = {
846                 # CONTENT OMITTED HERE
847                 # CHECK THE MANUAL IN THE ABOVE TEXT
848                 # OR CHECK THE SOURCE CODE FILE DIRECTLY
849                 # IT'S TOOOOOOOOO LONG TO BE PUT IN THE REPORT
850                 ......
851                 ......
852                 ......
853             }
854             if not args:
855                 # length is zero
856                 result = help_dict["MyShell"]
857             elif len(args) == 1:
858                 try:
859                     result = help_dict[args[0]]
860                 except KeyError as e:
```

```
861                 # raise HelpException(f"Cannot find manual entry for {args[0]}.
    {type(e).__name__}: {e}")
862                 log.warning("Cannot find help page in builtin dict, trying with
    man command")
863                 try:
864                     self.subshell(target="man", args=[args[0]], piping_in=False,
    piping_out=False)
865                 except CalledProcessException as e:
866                     raise HelpException(f"Cannot find help page for {args[0]}")
867                 return
868         else:
869             raise ArgException("Help takes one or zero argument.", {"type":
    "help"})
870 
871         # self.subshell(target="more", pipe=result, piping_in=True,
    piping_out=False)
872         self.subshell(target="less", pipe=result, piping_in=True,
    piping_out=False)
873 
874         return result
875 
876     def builtin_verbose(self, pipe="", args=[]):
877         # developer command: setting debug log printing level
878         supported = {
879             "-e": "ERROR",
880             "-w": "WARNING",
881             "-i": "INFO",
882             "-d": "DEBUG"
883         }
884         if len(args) > 1:
885             raise ArgException("This command takes zero or one argument.")
886         elif not args:
887             l = coloredlogs.get_level()
888             levels = [COLOR.BOLD(i) for i, k in
    coloredlogs.find_defined_levels().items() if k == l]
889             return "Current logging level: " + ", ".join(levels)
890         elif args[0] not in supported:
891             raise ArgException(f"Unrecognized argument: {args[0]}. We're
    supporting only {supported}.")
892 
893         coloredlogs.set_level(supported[args[0]])
894 
895     def subshell(self, pipe="", target="", args=[], piping_in=False,
    piping_out=False, io_control=False):
896         # 运行外部程序
897         # 根据需要调整输入输出
898 
899         # 是一个内部命令，通过调用subprocess的接口完成外部程序的调用
900         if not target:
901             raise EmptyException(f"Command \"{target}\" is empty", {"type":
    "subshell"})
902         to_run = [target] + args
903         log.debug(f"Runnning in subprocess: {COLOR.BOLD(to_run)}")
904         try:
905             log.debug(f"EXEC OI controller is: {COLOR.BOLD(str(self.input_file)
    + ' ' + str(self.output_file))}")
906             result = None
907             if io_control:
```

```
                piping_in = True
            # waits for the process to end
            # 若需要通过管道传递相关内容，则使用PIPE
            # ! windows中有些内置命令是无法通过subprocess调用的，例如type等cmd.exe内置命令
            p = subprocess.Popen(
                to_run,
                stdin=PIPE if piping_in else self.input_file,
                stdout=PIPE if piping_out else self.output_file,
                stderr=STDOUT, encoding="utf-8",
                env=dict(os.environ, PARENT=self.vars["SHELL"])
            )

            self.subp = p
            result, error = p.communicate(pipe)  # 如果我们选择不使用PIPE，这里传入空字符串也不会有任何影响
            if p.returncode != 0:
                log.warning("The subprocess is not returning zero exit code")
                raise CalledProcessException("None zero return code encountered")
        finally:
            # 我们使用try block的原因在于无论如何都要清空一下self.subp
            self.subp = None

        # 若通过subshell调用，则直接将结果以字符串形式返回
        return result

    def path(self):
        # callable PATH function，随着系统变量的改变而改变
        return self.vars["PATH"]

    def home(self):
        # callable HOME function，随着系统变量的改变而改变
        return self.vars['HOME']

    def cwd(self):
        # callable PWD function，随着系统变量的改变而改变
        cwd = os.getcwd()
        if cwd.startswith(self.home()):
            cwd = f"~{cwd[len(self.home())::]}"
        return cwd

    def user(self):
        # callable USER function，随着系统变量的改变而改变
        return getpass.getuser()

    def location(self):
        # callable LOCATION function，随着系统变量的改变而改变
        return platform.node()

    def prompt(self):
        # 返回将要打印到屏幕的命令提示符
        # 包含：用户@地点 当前目录 当前时间 提示符
        # ($CONDA_DEFAULT_ENV) $USER@location $PWD time("%H:%M:%S") $PS1
        prompt = f"{COLOR.BEIGE(self.user()+'@'+self.location())}
{COLOR.BOLD(COLOR.BLUE(self.cwd()))}
{COLOR.BOLD(datetime.datetime.now().strftime('%H:%M:%S'))}
{COLOR.BOLD(COLOR.YELLOW(self.vars['PS1'] if 'PS1' in self.vars else '$'))} "
```

```
960             # log.debug(repr(prompt))
961             try:
962                 conda = os.environ["CONDA_DEFAULT_ENV"]
963                 prompt = f"({conda}) {prompt}"
964             except KeyError:
965                 pass  # not a conda environment
966             return prompt
967 
968         def execute(self, command, pipe=""):
969             # 执行一个已经被格式化的命令
970             # 命令以dict形式传入，其中exec代表命令本身，args代表命令参数
971             # 还有一些其他控制参数，例如重定向或者管道操作
972             # 本函数还包含了对于exec命令的特殊处理（我们会提前处理重定向操作，而非交给命令本身，因此需要特殊化参数才可正常传入）
973             log.info(f"Executing command {COLOR.BOLD(command['exec'])}")
974             log.info(f"Arguments are {COLOR.BOLD(command['args'])}")
975 
976             # pipe或者重定向输入只能有一个
977             if command["pipe_in"] and command["redi_in"]:
978                 raise MultipleInputException("Redirection and pipe are set as input at the same time.")
979 
980             # piping_in/piping_out主要用于subshell的处理
981             command["piping_in"] = command["pipe_in"] or command["redi_in"]
982             command["piping_out"] = command["redi_out"] or command["pipe_out"]
983 
984             # 处理输入重定向
985             # to the command itself, it doesn't matter whether the input comes from a pipe or file
986             if command["redi_in"]:
987                 file_path = command["redi_in"]
988                 try:
989                     # the function open will automatically raise FileNotFoundError
990                     f = open(file_path, "r")
991                     pipe = f.read()
992                     f.close()
993                 except FileNotFoundError as e:
994                     raise FileNotFoundException(e, {"type": "redi_in"})
995 
996             # if we've specified input file in some thing
997             if self.input_file is not None:
998                 sys.stdin = self.input_file
999             # if we've specified output file in some thing
1000            if self.input_file is not None:
1001                sys.stdout = self.output_file
1002
1003            # actual execution
1004            result = ""
1005            try:
1006                if command["exec"] == "exec":
1007                    if len(command["args"]):
1008                        raise ArgException("exec command takes no argument")
1009                    # setting redi_files to arguments
1010                    # something might change
1011                    self.builtin_exec(args=[command["redi_in"], command["redi_out"], command["redi_append"]])
1012                elif command["exec"] in self.builtins.keys():
1013                    log.debug("This is a builtin command.")
```

```
1014                    # executing as static method, calling with self variable
1015                    result = self.builtins[command["exec"]](self, pipe=pipe,
    args=command['args'])
1016                else:
1017                    log.debug("This is not a builtin command.")
1018                    try:
1019                        result = self.subshell(pipe, command["exec"],
    command['args'], piping_in=command["piping_in"],
    piping_out=command["piping_out"], io_control=command["io_control"])
1020                    except FileNotFoundError as e:
1021                        raise FileNotFoundException(e, {"type": "subshell"})
1022            finally:
1023                # 我们使用try block的原因在于无论如何都要还原一下sys.stdin, sys.stdout
1024                sys.stdin = sys.__stdin__
1025                sys.stdout = sys.__stdout__
1026
1027            # 处理输出重定向
1028            if command['redi_out']:
1029                log.debug(f"User want to redirect the output to
    {COLOR.BOLD(command['redi_out'])}")
1030                try:
1031                    f = open(command["redi_out"], "a" if command["redi_append"] else
    "w")
1032                    f.write(result)
1033                    f.close()
1034                except FileNotFoundError as e:
1035                    raise FileNotFoundException(e, {"type": "redi_out",
    "redi_append": command["redi_append"]})
1036                return result
1037
1038            # 若用户需要进行管道操作，就不打印相关内容，直接以字符串返回获得的内容
1039            if command['pipe_out']:
1040                log.debug(f"User want to pipe the IO")
1041                return result
1042
1043            if result is not None:
1044                print(result, end="" if result.endswith("\n") or not len(result)
    else "\n", file=self.output_file)
1045            # return result # won't be used anymore
1046
1047        def command_prompt(self):
1048            # 命令提示符打印
1049            # note: readline quirk
1050            # strange error if we use input
1051            # the input and readline prompt seems to be counting color char as one
    of the line chars
1052            # well it turns out to be a quirk of readline
1053            # we've fixed it in COLOR.py
1054            try:
1055                command = input(self.prompt()).strip()
1056            except EOFError:
1057                # 在程序以交互模式运行，但是读入源非标准输入，就有可能在读完全部命令后得到
    EOFError
1058                # 我们选择在这种情况下退出交互模式
1059                # note: 这也意味着在普通的交互模式下输入Ctrl+D也会使shell退出
1060                print()
1061                log.warning("Getting EOF from command prompt, exiting...")
1062                return self.run_command("exit")
```

```
1063
1064            log.debug(f"Getting user input: {COLOR.BOLD(command)}")
1065            # 通过调用run_command来执行相关内容
1066            result = self.run_command(command)
1067            return result
1068
1069        @staticmethod
1070        def run_command_wrap(count, shell, args, job, queue, jobs, status_dict):
1071            # 用于后台程序管理
1072            # 我们不仅可以后台运行外部命令，程序自身命令也可以后台执行
1073            # 且管道，重定向等操作都由MyShell执行，这与直接刷出一个新的subprocess完全不同
1074            # 我们没有在这种情况下借用已有shell的功能
1075            log.debug(f"Wrapper [{count}] called with {COLOR.BOLD(f'{shell} and
    {args}')}")
1076
1077            # ! DOESN"T WORK ON WINDOWS!
1078            # ! WILL CREATE ZOMBIE PROCESS!
1079
1080            # 为了在信号处理过程中正确杀死自身进程
1081            my_pid = os.getpid()
1082
1083            # signal handler，遇到signal.SIGTERM后杀死可能正在运行的子进程
1084            def exit_subp(sig, frame):
1085                if shell.subp is not None:
1086                    log.warning("Killing a still running subprocess ... ")
1087                    os.kill(shell.subp.pid, signal.SIGKILL)
1088                log.debug(f"Getting signal: {COLOR.BOLD(sig)}")
1089                log.debug(f"Are you: {COLOR.BOLD(signal.SIGTERM)}")
1090                if sig == signal.SIGTERM:
1091                    # 杀死自身进程
1092                    log.warning(f"Terminating job [{count}] handler process by
    signal ... ")
1093                    os.kill(my_pid, signal.SIGKILL)
1094
1095            # 注册信号处理器
1096            # note: multiprocessin.Process.daemon = True时，若父进程退出，该信号会被触发
1097            signal.signal(signal.SIGTERM, exit_subp)
1098
1099            # note: 正文
1100            # 通过Wrapper来控制内部命令的suspension
1101            if sys.stdin is not None:
1102                sys.stdin.close()
1103            # stdin的文件号
1104            stdin = open(0)
1105            buffer = stdin.detach()
1106            wrapper = MyShell.StdinWrapper(queue, count, job, status_dict, buffer)
1107            sys.stdin = wrapper
1108
1109            try:
1110                # 真正的执行过程
1111                # 详见run_command
1112                # 包括解析命令/执行命令/错误处理
1113                # 此时不会执行后台逻辑，跳过parse阶段
1114                shell.run_command(args, io_control=True)
1115            finally:
1116                # 正常情况下不会出现raise的情况
1117                # 无论如何，都要退出当前的job
1118                print(shell.job_status_fmt(count, "finished", job))
```

```
1119                queue.put("dummy")
1120                del jobs[count]
1121
1122        def term_all(self):
1123            # 杀死jobs数组中所有的剩余任务
1124            jobs = self.jobs
1125            for job in jobs:
1126                os.kill(self.process[job].pid, signal.SIGTERM)
1127                # 我们会将job相关信息直接打印到屏幕上
1128                print(self.job_status_fmt(job, "terminated", jobs[job]))
1129
1130        def run_command(self, command, io_control=False):
1131            # 解析命令，执行命令，错误处理，后台任务
1132            try:
1133                # commands是一个command数组，包含具体的方便读取的命令格式信息
1134                # 由于存在PIPE调用的可能性，我们对commands创建了数组
1135                # parse会将命令字符串解释为完整的命令
1136                commands, is_bg = self.parse(command)
1137                if is_bg:
1138                    # ! changes made in subprocess is totally within the subprocess
    only
1139
1140                    # job_counter实际上是一个函数
1141                    # 会获取当前没有被使用掉的最小的job编号，从0开始
1142                    str_cnt = self.job_counter
1143
1144                    # 创建相关内容
1145                    self.jobs[str_cnt] = command
1146                    self.queues[str_cnt] = Queue()
1147                    self.status_dict[str_cnt] = "running"
1148
1149                    # 备份相关内容以便deepcopy时删除
1150                    queues_bak = self.queues
1151                    process_bak = self.process
1152                    jobs_bak = self.jobs
1153                    status_dict_bak = self.status_dict
1154                    input_bak = self.input_file
1155                    output_bak = self.output_file
1156
1157                    # note: 为了规避一些pickle error，我们会删除这里的：
1158                    # thread_lock, multiprocessing.Manager, multiproessing.Queue,
    Manager.dict, io.TextWrapper
1159                    del self.queues
1160                    del self.process
1161                    del self.jobs
1162                    del self.status_dict
1163                    del self.input_file
1164                    del self.output_file
1165
1166                    # 获取一个deepcopy的shell以供后台程序执行
1167                    clean_self = copy.deepcopy(self)
1168
1169                    # 填充默认内容
1170                    # ! 我们默认后台程序不会尝试再次构建后台的后台指令
1171                    clean_self.queues = {}
1172                    clean_self.jobs = {}
1173                    clean_self.process = {}
1174                    clean_self.status_dict = {}
```

```
1175                    clean_self.input_file = None
1176                    clean_self.output_file = None
1177
1178                    # 还原到deepcopy以前的状态
1179                    self.queues = queues_bak
1180                    self.process = process_bak
1181                    self.jobs = jobs_bak
1182                    self.status_dict = status_dict_bak
1183                    self.input_file = input_bak
1184                    self.output_file = output_bak
1185
1186                    # 使用deepcopy初来的clean_self来构建新的进程
1187                    p = Process(target=self.run_command_wrap, args=(str_cnt,
    clean_self, command[0:-1], self.jobs[str_cnt], self.queues[str_cnt], self.jobs,
    self.status_dict), name=command)
1188
1189                    # 添加到进程管理中，方便kill命令等的执行
1190                    self.process[str_cnt] = p
1191
1192                    # ! so the multiprocessing process won't actually be totally
    gone if the main process is still around
1193                    # but it will be terminated if demanded so (ps command can see
    it as <defunc>, but not a zombie)
1194                    # 保证主进程退出后，下面的小进程也会退出
1195                    p.daemon = True
1196
1197                    # 开始运行
1198                    p.start()
1199                    log.debug(f"We've spawned the job in a Process for command:
    {COLOR.BOLD(p.name)}")
1200                else:
1201                    # 一般命令的执行
1202                    result = None  # so that the first piping is directly from stdin
1203                    if io_control:
1204                        for command in commands:
1205                            command["io_control"] = True
1206                    else:
1207                        for command in commands:
1208                            command["io_control"] = False
1209                    for cidx, command in enumerate(commands):
1210                        result = self.execute(command, pipe=result)
1211                        # log.debug(f"Getting result: {COLOR.BOLD(result)}")
1212            except ExitException as e:
1213                # 我们通过异常来处理程序的退出命令
1214                # 包括quit和exit
1215                # 某种意义上Ctrl+D代表的EOF也是一种推出指令
1216                log.debug("User is exiting ... ")
1217                log.debug(f"Exception says: {e}")
1218                log.info("Bye")
1219                self.term_all()
1220                return True
1221            except EmptyException as e:
1222                # EmptyException较为特殊，需要单独处理
1223                # 因为错误等级会根据type不同而变换
1224                log.debug("The command is empty ... ")
1225                log.info(f"Exception says: {e}")
1226                if e.errors["type"] == "pipe":
1227                    log.error(f"Your pipe is incomplete. {e}")
```

```
1228                elif e.errors["type"] == "subshell":
1229                    log.warning(f"Your command is empty. Did you use an empty var?
    {e}")
1230                elif e.errors["type"] == "empty":
1231                    log.info(f"Your command is empty. {e}")
1232            except MyShellException as e:
1233                # this means that we've got a none zero return code / execution
    failure
1234                error_cmd = command['exec'] if isinstance(command, dict) else
    command
1235                line_end = "\n"
1236                log.error(f"Cannot successfully execute command \"{error_cmd}\".
    Exception is: {line_end}{COLOR.BOLD(type(e).__name__ + ': ' + str(e) + line_end
    + 'Extra info: ' + str(e.errors))}")
1237            except Exception as e:
1238                log.error(f"Unhandled error. {traceback.format_exc()}")
1239            finally:
1240                # 同步status_dict, queues, jobs
1241                self.cleanup_jobs()  # always clean up
1242
1243            # 不返回退出指令
1244            return False
1245
1246        def parse(self, command):
1247            # 命令解释器
1248
1249            # quote函数会处理引号/变量替换/特殊变量转义/~替换
1250            inputs = self.quote(command)  # splitting by whitespace and processing
    varibles, quotes
1251
1252            # 将命令分解，进行piping调用
1253            # splitting command of pipling
1254            commands = []
1255            command = []
1256            for word in inputs:
1257                if word != "|":
1258                    command.append(word)
1259                else:
1260                    commands.append(command)
1261                    command = []
1262            commands.append(command)
1263
1264            # 对每一个个别的命令进行解析
1265            is_bg = False  # 是否后台执行是整个命令的设置，只需要一个
1266            parsed_commands = []
1267
1268            for cidx, command in enumerate(commands):
1269                parsed_command = self.parsed_clean()
1270                # 可能用户只是敲了一下回车
1271                # 也可能用户的某一个管道环节是空的：这是不允许的
1272                # 我们通过type传递相关信息
1273                if not len(command):
1274                    raise EmptyException(f"Command at position {cidx} is empty",
    {"type": "pipe" if len(commands) > 1 else "empty"})
1275                parsed_command["pipe_in"] = (cidx > 0)
1276                parsed_command["pipe_out"] = (cidx != len(commands)-1)
1277                index = 0
1278                while index < len(command):
```

```
1279                 if not index:  # this should have a larger priority
1280                     # 整个命令的第一个词汇是执行目标
1281                     parsed_command["exec"] = command[index]
1282                 elif command[index] == "<":
1283                     # 输入重定向
1284                     if index == len(command) - 1:
1285                         raise SetPairUnmatchedException("Cannot match
     redirection input file with < sign.", {"type": "redi"})
1286                     parsed_command["redi_in"] = command[index+1]
1287                     index += 1
1288                 elif command[index] == ">":
1289                     # 输出重定向
1290                     if index == len(command) - 1:
1291                         raise SetPairUnmatchedException("Cannot match
     redirection output file with > sign.", {"type": "redi"})
1292                     parsed_command["redi_out"] = command[index+1]
1293                     index += 1
1294                 elif command[index] == ">>":
1295                     # 输出重定向：添加型
1296                     if index == len(command) - 1:
1297                         raise SetPairUnmatchedException("Cannot match
     redirection output file with >> sign.", {"type": "redi"})
1298                     parsed_command["redi_out"] = command[index+1]
1299                     parsed_command["redi_append"] = True
1300                     index += 1
1301                 elif command[index] == "&":
1302                     if index != len(command)-1 or cidx != len(commands) - 1:
1303                         # & not at the end
1304                         raise UnexpectedAndException("Syntax error, unexpected &
     found")
1305 
1306                     # todo: communication
1307                     is_bg = True
1308                     log.info("User want this to run in background.")
1309                 else:
1310                     parsed_command["args"].append(command[index])
1311                 index += 1
1312             parsed_commands.append(parsed_command)
1313 
1314         return parsed_commands, is_bg
1315 
1316     def quote(self, command, quote=True):
1317         # 可以被递归调用的引号处理功能
1318         # command should already been splitted by word
1319         # mark: this structure makes it possible that the arg is keep in one
     place if using quote
1320         # by not brutally expanding on every possible environment variables
1321         command = command.split()
1322         # 清除所有的注释
1323         # note: 就像一般的shell，注释前面要有一个空格
1324         comment = [i for i, v in enumerate(command) if v.startswith("#")]
1325         if len(comment):
1326             command = command[0:comment[0]]
1327 
1328         quote_stack = []
1329         index = 0
1330         while index < len(command):
1331 
```

```
1332                # 这一个if-else代码块保证所有内容被解析过一次，且仅解析一次，调用self.expand
    函数
1333                # 并且若解析后的内容有引号，也不会影响程序的正常执行，因为我们是在处理引号后解
    析变量的
1334                # note: 也就是说如果你的变量中存在着可以被解析为变量的内容，它们不会被解析，以
    防止无限递归解析
1335                if command[index].count("\"") >= 1:
1336                    # should remove all quotes here
1337                    quote_count = command[index].count("\"")
1338                    log.debug("Trying to remove quote ... ")
1339                    splitted = command[index].split("\"")
1340                    # there were not space at the beginning
1341                    # expand函数会进行变量和~的替换
1342                    # 我们会将读入的内容按照引号数量拆分，然后进行解析
1343                    # 最后合并成一个长字符串
1344                    # 并通过quote_count来和下面的代码沟通
1345                    command[index] = "".join([self.expand(split) for split in
    splitted])
1346                else:
1347                    command[index] = self.expand(command[index])
1348                    quote_count = 0
1349
1350                if quote_count % 2:  # previous quote_count
1351                    # 引号分词中出现了空格
1352                    if quote_stack:
1353                        # except the last char
1354                        quote_stack.append(command[index])
1355                        # recursion to process $ and "" in the already processed ""
1356                        command[index] = " ".join(quote_stack)
1357                        quote_stack = ""
1358                        # index should continue to be added in the end
1359                    else:
1360                        log.debug("Trying to match quote ... ")
1361                        quote_stack.append(command[index])
1362                        if index == len(command)-1:
1363                            raise QuoteUnmatchedException("Cannot match the quote
    for the last argument")
1364                        command = command[0:index] + command[index+1::]
1365                        index -= 1
1366                elif quote_stack:
1367                    # 引号分词没有空格，加入到上一个quote组中
1368                    # no quote but
1369                    quote_stack.append(command[index])
1370                    if index == len(command)-1:
1371                        raise QuoteUnmatchedException("Cannot match the quote for a
    series of arguments")
1372                    command = command[0:index] + command[index+1::]
1373                    index -= 1  # index should stay the same
1374                index += 1
1375
1376            return [i.replace("\\~", "~").replace("\\$", "$").replace("\\#", "#")
    for i in command]
1377
1378        def expand(self, string):
1379            def get_key(key):
1380                # 第一个字符$不能被用于寻找变量
1381                key = key[1::]
1382                log.debug(f"Trying to get varible {COLOR.BOLD(key)}")
```

```
1383                var = self.vars[key]
1384                # splitting the expanded command since it might contain some
    information
1385                return var
1386
1387            def get_home(key):
1388                log.debug(f"GETTING HOME SIGN: {COLOR.BOLD(key)}")
1389                if key == "~":
1390                    return self.home()
1391                else:
1392                    return key
1393
1394            # 通过调用regex的替换命令（其实是我们自己实现的）
1395            # ! re.sub在处理utf-8字符串时有难以预料的错误
1396            # ! if you use re.sub on windows, strange things can happen
1397            # *nix可以正常运行，但Windows下报错
1398            string = self.sub_re(r"(?<!\\)\$\w+", string, get_key)
1399            string = self.sub_re(r"(?<!\\)~", string, get_home)
1400
1401            return string
1402
1403        @staticmethod
1404        def sub_re(pattern, string, method):
1405            # 根据regex寻找匹配子串，然后通过调用method函数进行替换的静态方法
1406
1407            # 所有的匹配结果
1408            var_list = [(m.start(0), m.end(0)) for m in re.finditer(pattern,
    string)]
1409
1410            # 拆分为替换后的内容
1411            str_list = []
1412            prev = [0, 0]
1413            for start, end in var_list:
1414                str_list.append(string[prev[1]:start])
1415                str_list.append(string[start:end])
1416                prev = [start, end]
1417            str_list.append(string[prev[1]::])
1418
1419            # log.debug(f"The splitted vars are {COLOR.BOLD(str_list)}")
1420
1421            # 对于每一个匹配成功的字串，进行method(key)的调用
1422            for i in range(1, len(str_list), 2):
1423                str_list[i] = method(str_list[i])
1424                # to result in index staying the same
1425
1426            string = "".join(str_list)
1427            return string
1428
1429        def parsed_clean(self):
1430            # 返回一个干净的指令
1431            parsed_command = {
1432                "exec": "",
1433                "args": [],
1434                "pipe_in": False,
1435                "pipe_out": False,
1436                "redi_in": None,
1437                "redi_out": None,
1438                "redi_append": False,
```

```
1439            }
1440            return parsed_command
1441
1442
1443    if __name__ == "__main__":
1444        # cat < dummy.mysh | wc > /dev/tty | echo "zy" > result.out | sha256sum | tr
    -d " -" >> result.out | wc | cat result.out | wc | cat result.out
1445        # test ! -z "" -a ( -n "1" -o 1 -ge 1 ) -o 2 -ne 1 # False, -a -o from right
    to left
1446        # test ( ! -z "" -a ( -n "1" -o 1 -ge 1 ) ) -o 2 -ne 1 # True
1447        # ./MyShell.py -w dummy.mysh -a foo bar foobar hello world linux linus
    PyTorch CS231n
1448
1449        # 调用此脚本时候传入-h以查看帮助
1450        parser = argparse.ArgumentParser(description='MyShell by
    xudenden@gmail.com')
1451        parser.add_argument('f', metavar='F', type=str, nargs='?', help='the batch
    file to be executed')
1452        parser.add_argument('-a', metavar='A', type=str, nargs='*', help='command
    line arguments to batch file')
1453        parser.add_argument('-e', help='enable error level debugging info log',
    action='store_true')
1454        parser.add_argument('-w', help='enable warning level debugging info log',
    action='store_true')
1455        parser.add_argument('-i', help='enable info level debugging info log',
    action='store_true')
1456        parser.add_argument('-d', help='enable debug(verbose) level debugging info
    log', action='store_true')
1457
1458        args = parser.parse_args()
1459        # args为MyShell的命令行参数
1460        # 处理MyShell的命令行参数
1461        if args.e:
1462            coloredlogs.set_level("ERROR")
1463        if args.w:
1464            coloredlogs.set_level("WARNING")
1465        if args.i:
1466            coloredlogs.set_level("INFO")
1467        if args.d:
1468            coloredlogs.set_level("DEBUG")
1469
1470        # 命令行参数的第一个为当前脚本的路径（脚本模式）·或MyShell的路径（交互模式）
1471        cmd_args = [os.path.abspath(args.f) if args.f else
    os.path.abspath(__file__)]
1472        if args.a is not None:
1473            cmd_args += args.a
1474        myshell = MyShell(cmd_args=cmd_args)
1475        log.debug(f"Getting user command line argument(s): {COLOR.BOLD(args)}")
1476
1477        # ! 一次只准执行一个脚本
1478        if args.f:
1479            try:
1480                # ! using utf-8
1481                f = open(args.f, encoding="utf-8")  # opending the file specified
1482                for line in f:
1483                    line = line.strip()
1484                    result = myshell.run_command(line)  # execute line by line
1485                    if result:  # the execution of the file should be terminated
```

```
                    break
            myshell.run_command("exit")  # call exit at the end of the shell execution
        except FileNotFoundError as e:
            log.error(f"Cannot find the file specified for batch processing: \"{args.f}\". {e}")
    else:
        # 直接使用交互模式
        myshell()

```