

系统开发工具基础课程实验报告

姓名：张誉馨

2024 年 9 月 17 日

目录

1	练习内容和结果	1
1.1	调试及性能分析	1
1.2	元编程	6
1.3	大杂烩	7
1.4	pytorch	10
2	解题感悟	10
3	github链接	11

1 练习内容和结果

1.1 调试及性能分析

1.使用 Linux 上的 `journalctl` 或 macOS 上的 `log show` 命令来获取最近一天中超级用户的登录信息及其所执行的指令。如果找不到相关信息，您可以执行一些无害的命令，例如 `sudo ls` 然后再次查看。

```
islovcvm:~$ journalctl | grep sudo
3月 10 15:56:49 islovcvm sshd[43988]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ;
COMMAND=/usr/bin/chown root file.txt
3月 10 15:56:49 islovcvm sshd[43988]: pam_unix(sshd:session): session opened for user root by (l
id=0)
3月 10 15:56:49 islovcvm sshd[43988]: pam_unix(sshd:session): session closed for user root
3月 10 16:01:17 islovcvm sshd[44000]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ;
COMMAND=/usr/bin/chgrp -R 目录所有者 目录名称
3月 10 16:01:17 islovcvm sshd[44000]: pam_unix(sshd:session): session opened for user root by (l
id=0)
3月 10 16:01:17 islovcvm sshd[44000]: pam_unix(sshd:session): session closed for user root
3月 10 19:32:51 islovcvm sshd[44679]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ;
COMMAND=/usr/bin/apt install sysstat
3月 10 19:32:51 islovcvm sshd[44679]: pam_unix(sshd:session): session opened for user root by (l
id=0)
3月 10 19:32:58 islovcvm sshd[44679]: pam_unix(sshd:session): session closed for user root
3月 21 19:26:29 islovcvm sshd[63381]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ;
COMMAND=/usr/bin/passwd root
3月 21 19:26:29 islovcvm sshd[63381]: pam_unix(sshd:session): session opened for user root by (l
id=0)
3月 21 19:26:52 islovcvm sshd[63381]: pam_unix(sshd:session): session closed for user root
3月 21 19:27:11 islovcvm sshd[63383]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ;
COMMAND=/usr/bin/passwd root
3月 21 19:27:11 islovcvm sshd[63383]: pam_unix(sshd:session): session opened for user root by (l
id=0)
3月 21 19:27:18 islovcvm sshd[63383]: pam_unix(sshd:session): session closed for user root
3月 30 18:23:05 islovcvm sshd[5025]: ouc : TTY=pts/0 ; PWD=/home/ouc/Desktop ; USER=root ;
COMMAND=/usr/bin/apt-get update
3月 30 18:23:05 islovcvm sshd[5025]: pam_unix(sshd:session): session opened for user root by (u
id=0)
3月 30 18:23:15 islovcvm sshd[5025]: pam_unix(sshd:session): session closed for user root
```

图 1: 获取最近一天中超级用户的登录信息及其所执行的指令

2.学习这份 `pdb` 实践教程并熟悉相关的命令。更深入的信息您可以参考这份教程。



图 2: pdb教程

3.安装 `shellcheck` 并尝试对下面的脚本进行检查。

```
#!/bin/sh
## Example: a typical script with several problems
for f in $(ls *.m3u)
do
    grep -qi hq.*mp3 $f \
    && echo -e 'Playlist $f contains a HQ file in mp3 format'
done
```

图 3: 待检查脚本

```
mac@shellcheck:~$ sudo apt-get install shellcheck
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  shellcheck
0 upgraded, 1 newly installed, 0 to remove and 52 not upgraded.
Need to get 2,383 kB of archives.
After this operation, 17.6 MB of additional disk space will be used.
Get:1 https://mirrors.cloud.tencent.com/ubuntu focal/universe amd64 shellcheck amd64 0.7.0-2build1
2 [2,383 kB]
Fetched 2,383 kB in 1s (4,592 kB/s)
Selecting previously unselected package shellcheck.
(Reading database ... 194332 files and directories currently installed.)
Preparing to unpack .../shellcheck_0.7.0-2build1_amd64.deb ...
Unpacking shellcheck (0.7.0-2build1) ...
Setting up shellcheck (0.7.0-2build1) ...
Processing triggers for man-db (2.9.1-1) ...
```

图 4: 安装shellcheck

4.我们经常会遇到的情况是某个我们希望去监听的端口已经被其他进程占用了。让我们通过进程的PID查找相应的进程。首先执行 `python -m http.server 4444` 启动一个最简单的 web 服务器来监听 4444 端口。在另外一个终端中，执行 `lsof | grep LISTEN` 打印出所有监听端口的进程及相应的端口。找到对应的 PID 然后使用 `kill <PID>` 停止该进程。

5.检查使用的python版本

你可以在你的终端上运行如下的命令：

```
python --version
```

6.pdb教程中以游戏形式引导

我们已经讨论了调试器的功能，现在到了先看看它是如何工作的时候了。首先，如果你还没有克隆这个仓库的话要先克隆。如果你还没有安装git的话，我建议你尝试安装并使用它（或者是其他的源代码工具），您可以参考这里来安装git。在您安装好git之后，在您的终端运行以下命令来克隆仓库：

```
git clone https://github.com/spiside/pdb-tutorial
```

7.接上一问，在终端输入以下命令来运行这个程序：

```
python main.py
```

8.接上一问，是时候到了使用 python 自带的调试器pdb的时候了。这个调试器包含在 python 的标准库中，我们就像使用任何一个 python 库一样来使用它。首先，我们应先加载pdb模块，然后调用它的方法在程序中设

```
ouc@islouc-vm:~$ vim myscript.sh
Error detected while processing /home/ouc/.vimrc:
line 15:
E492: Not an editor command: 语法高亮
line 48:
E492: Not an editor command: 然后把它保存到 ~/.vimrc! Comments in Vimscript start with a " ".
Press ENTER or type command to continue
ouc@islouc-vm:~$ shellcheck myscript.sh

In myscript.sh line 3:
for f in $(ls *.mp3)
----- SC2045: Iterating over ls output is fragile. Use glob.
... SC2035: Use /*glob* or /*glob* so names with dashes won't become options.

In myscript.sh line 5:
grep -q1 hq.*mp3 $f \
----- SC2062: Quote the grep pattern so the shell won't interpret it.
... SC2066: Double quote to prevent globbing and word splitting.

Did you mean:
grep -q1 hq.*mp3 "$f" \

In myscript.sh line 6:
&& echo -e 'Playlist $f contains a HQ file in mp3 format'
----- SC2039: In POSIX sh, echo flags are undefined.
... SC2016: Expressions don't expand in single quotes, use double quotes for that.

For more information:
https://www.shellcheck.net/wiki/SC2045 -- Iterating over ls output is fragil...
https://www.shellcheck.net/wiki/SC2039 -- In POSIX sh, echo flags are undef...
https://www.shellcheck.net/wiki/SC2062 -- Quote the grep pattern so the she...
```

图 5: 对脚本进行检查

```
ouc@islouc-vm:~$ python -m http.server 444
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3
^
ouc@islouc-vm:~$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
^C
Keyboard interrupt received, exiting.
ouc@islouc-vm:~$ python -m http.server 4444
Command 'python' not found, did you mean:
  command 'python3' from deb python3
  command 'python' from deb python-is-python3
ouc@islouc-vm:~$ python3 -m http.server 4444
Serving HTTP on 0.0.0.0 port 4444 (http://0.0.0.0:4444/) ...
```

图 6: 启动web服务器来监听端口

置用于调试的程序断点。传统的方法是将加载和调用放在想要程序暂停运行的位置。这个是会用到的完整声明语句：

```
import pdb; pdb.set_trace()
```

set_trace()方法在被调用的地方设置了一个程序断点。让我们来打开main.py文件并在第8行添加程序断点：

```
file: main.py
```

```
from dicegame.runner import GameRunner
```

```
def main():
```

```
    print("Add the values of the dice")
```

```
    print("It's really that easy")
```

```
    print("What are you doing with your life.")
```

```
    import pdb; pdb.set_trace() # add pdb here
```

```
    GameRunner.run()
```

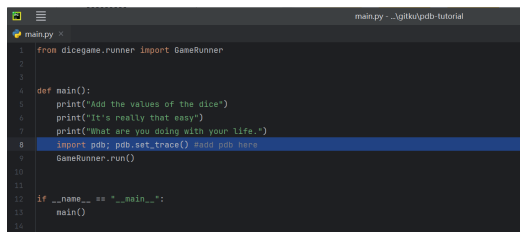



图 11: 在main.py中添加程序断点



图 12: 再次运行main.py

让我们称之为 return命令并到达函数的末尾。

1.2 元编程

15.大多数的 makefiles 都提供了一个名为 clean 的构建目标，这并不是说我们会生成一个名为clean的文件，而是我们可以使用它清理文件，让 make 重新构建。您可以理解为它的作用是“撤销”所有构建步骤。在上面的 makefile 中为paper.pdf实现一个clean 目标。您需要构建phony。您也许会发现 git ls-files 子命令很有用。其他一些有用的 make 构建目标可以在这里找到；

16.指定版本要求的方法很多，学习一下 Rust的构建系统的依赖管理。大多数的包管理仓库都支持类似的语法。对于每种语法(尖号、波浪号、通配符、比较、乘积)，构建一种场景使其具有实际意义；

17.Git 可以作为一个简单的 CI 系统来使用，在任何 git 仓库中的 .git/hooks 目录中，您可以找到一些文件（当前处于未激活状态），它们的作用和脚本一样，当某些事件发生时便可以自动执行。请编写一个pre-commit 钩子，当执行make命令失败后，它会执行 make paper.pdf 并拒绝您的提交。这样做可以避免产生包含不可构建版本的提交信息；

修改.git/hooks 目录下面的pre-commit.sample文件并将其命名为pre-commit
if ! make ; then

```

$ python main.py
Add the values of the dice
It's really that easy
What are you doing with your life.
> e:\gitku\pdb-tutorial\main.py(9):main()
-> gameRunner.run()
(Pdb) help list
usage: list([first[, last] | .])

List source code for the current file, without arguments,
list 11 lines around the current line or continue the previous
listing, with . as argument, list 11 lines around the current
line, with one argument, list 11 lines starting at that line.
With two arguments, list the given range; if the second
argument is less than the first, it is a count.

The current line in the current frame is indicated by ">".
If an exception is being debugged, the line where the
exception was originally raised or propagated is indicated by
">>". If it differs from the current line.

(Pdb)

```

图 13: 学习list功能

```

(Pdb) s
--call--
> e:\gitku\pdb-tutorial\dicegame\runner.py(21):run()
-> @classmethod
(Pdb) l
16         total = 0
17         for die in self.dice:
18             total += 1
19         return total
20
21     -> @classmethod
22     def run(cls):
23         # Probably counts wins or something.
24         # Great variable name, 10/10.
25         c = 0
26         while True:

```

图 14: 调用step后调用list

```

echo "build failed, commit rejected"
exit 1
fi

```

1.3 大杂烩

18.创建键位映射。一个很常见的配置是修改键位映射。通常这个功能由在计算机上运行的软件实现。当某一个按键被按下，软件截获键盘发出的按键事件（keypress event）并使用另外一个事件取代。比如：将 Caps Lock 映射为 Ctrl 或者 Escape：Caps Lock 使用了键盘上一个非常方便的位置而它的功能却很少被用到，所以非常推荐这个修改；将 PrtSc 映射为播放/暂停：大部分操作系统支持播放/暂停键；交换 Ctrl 和 Meta 键（Windows 的徽标键或者 Mac 的 Command 键）。也可以将键位映射为任意常用的指令。软件监听到特定的按键组合后会运行设定的脚本。

打开一个新的终端或者浏览器窗口；输出特定的字符串，比如：一个超长邮件地址或者 MIT ID；使计算机或者显示器进入睡眠模式。甚至更复杂的修改也可以通过软件实现：

映射按键顺序，比如：按 Shift 键五下切换大小写锁定；区别映射单点


```

(Pdb) s
> e:\gitku\pdb-tutorial\dicegame\runner.py(25)run()
-> c = 0
(Pdb) l
20
21
22     @classmethod
23     def run(cls):
24         # Probably counts wins or something.
25         # Great variable name, 10/10.
26         c = 0
27         while True:
28             runner = cls()
29             print("Round {}".format(runner.round))
30

```

图 15: 再次执行step后调用list

```

(Pdb) n
> e:\gitku\pdb-tutorial\dicegame\runner.py(26)run()
-> while True:
(Pdb) l
21
22     @classmethod
23     def run(cls):
24         # Probably counts wins or something.
25         # Great variable name, 10/10.
26         c = 0
27         while True:
28             runner = cls()
29             print("Round {}".format(runner.round))
30             for die in runner.dice:
31

```

图 16: 输入next命令后调用list

和长按，比如：单点 Caps Lock 映射为 Escape，而长按 Caps Lock 映射为 Ctrl；对不同的键盘或软件保存专用的映射配置。

19.守护进程。大部分计算机都有一系列在后台保持运行，不需要用户手动运行或者交互的进程。这些进程就是守护进程。以守护进程运行的程序名一般以 d 结尾，比如 SSH 服务端 sshd，用来监听传入的 SSH 连接请求并对用户进行鉴权。

Linux 中的 systemd (the system daemon) 是最常用的配置和运行守护进程的方法。运行 systemctl status 命令可以看到正在运行的所有守护进程。这里面有很多可能你没有见过，但是掌管了系统的核心部分的进程：管理网络、DNS 解析、显示系统的图形界面等等。用户使用 systemctl 命令和 systemd 交互来 enable (启用)、disable (禁用)、start (启动)、stop (停止)、restart (重启)、或者 status (检查) 配置好的守护进程及系统服务。

20.FUSE。现在的软件系统一般由很多模块化的组件构建而成。你使用的操作系统可以通过一系列共同的方式使用不同的文件系统上的相似功能。比如当你使用 touch 命令创建文件的时候，touch 使用系统调用 (system call) 向内核发出请求。内核再根据文件系统，调用特有的方法来创建文件。这里的问题是，UNIX 文件系统在传统上是以内核模块的形式实现，导致只有内核可以进行文件系统相关的调用。


```

paper.pdf: paper.tex plot-data.png
        pdflatex paper.tex

plot-%.png: %.dat plot.py
        ./plot.py -i $*.dat -o $@

.PHONY: clean
clean:
        rm *.pdf *.aux *.log *.png
        #git ls-files -o | xargs rm -f

```

图 19: 编写makefile

Specifying Dependencies

Your crates can depend on other libraries from crates.io or other registries, `git` repositories, or subdirectories on your local file system. You can also temporarily override the location of a dependency — for example, to be able to test out a bug fix in the dependency that you are working on locally. You can have different dependencies for different platforms, and dependencies that are only used during development. Let's take a look at how to do each of these.

Specifying dependencies from crates.io

Cargo is configured to look for dependencies on crates.io by default. Only the name and a version string are required in this case. In the [cargo guide](#), we specified a dependency on the `time` crate:

```
[dependencies]
time = "0.1.12"
```

The string `"0.1.12"` is a version requirement. Although it looks like a specific version of the `time` crate, it actually specifies a *range* of versions and allows [SemVer](#) compatible updates. An update is allowed if the new version number does not modify the left-most non-zero number in the major, minor, patch grouping. In this case, if we ran `cargo update time`, cargo should update us to version `0.1.13` if it is the latest `0.1.x` release, but would not update us to `0.2.0`. If instead we had specified the version string as `1.0`, cargo should update to `1.1` if it is the latest `1.y` release, but not `2.0`. The version `0.0.x` is not considered compatible with any other version.

图 20: 学习rust构建系统的依赖管理

1.4 pytorch

24.张量是一种特殊的数据结构，与数组和矩阵非常相似。在PyTorch中，我们使用张量对模型的输入和输出以及模型的参数进行编码。

张量类似于数字Py但张量可以在GPU或其他硬件加速器上运行。事实上，张量和NumPy数组通常可以共享相同的底层内存，从而无需复制数据（请参阅带NumPy的桥）。张量也针对自动微分进行了优化。如果熟悉darrays，就可以轻松使用Tensor API。

2 解题感悟

通过这次实验我了解了调试和性能分析、元编程、大杂烩以及pytorch。学习了一些pdb命令，比如l(ist)、s(tep)、n(ext)、b(reak)、r(eturn)等，他们的功能可以根据字面意思来理解，如果想详细了解可以help后加命令。

```
l.3 b
    egin{document}
! ==> Fatal error occurred, no output PDF file produced!
Transcript written on paper.log.
make: *** [paper.pdf] Error 1
build failed, commit rejected
x> /tmp/missing/build p master+
```

图 21: 修改.git/hooks目录下的pre-commit.sample


Type	Or	... to Get
Italic	..Italic..	<i>Italic</i>
Bold	...Bold...	Bold
# Heading 1	Heading 1 ~~~~~	Heading 1
## Heading 2	Heading 2 ~~~~~	Heading 2
[Link](http://a.com)	[Link][1] [1]: http://b.org	Link
![Image](http://url/a.png)	![Image][1] [1]: http://url/b.jpg	
> Blockquote		Blockquote
* List	- List	• List
* List	- List	• List
* List	- List	• List

图 22: markdown的一些使用指南

了解了安装并使用shellcheck来检查脚本，还了解了创建键位映射、守护进程、FUSE、备份、markdown等。

3 github链接

<https://github.com/zyx-cyber/coursecontent.git>

初始化张量

张量可以用各种方式初始化。请看以下示例：

直接从数据

张量可以直接从数据中创建，自动推断数据类型。

```
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
```

从NumPy数组

张量可以从NumPy数组创建（反之亦然参见[带NumPy的桥](#)）。

```
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
```

从另一张量：

除非显式重写，否则新张量保留参数张量的属性（形状、数据类型）。

```
x_ones = torch.ones_like(x_data) # retains the properties of x_data
print(f"ones Tensor: \n {x_ones} \n")

x_rand = torch.rand_like(x_data, dtype=torch.float) # overrides the datatype of x_data
print(f"Random Tensor: \n {x_rand} \n")
```

图 23: pytorch中张量的初始化