

# 系统工具开发基础实验

曹瑜 22020007007

September 12, 2024

## 1 实验内容

命令行环境

python 基础

python 视觉应用

## 2 实验目的

(1) 学习如何同时执行多个不同的进程并追踪它们的状态、如何停止或暂停某个进程以及如何使进程在后台运行。

(2) 学习一些能够改善 shell 及其他工具的工作流的方法，主要是通过定义别名或基于配置文件对其进行配置来实现的。这些方法都可以节省大量的时间。

## 3 实验步骤

### 3.1 任务控制

shell 会使用 UNIX 提供的信号机制执行进程间通信。当一个进程接收到信号时，它会停止执行、处理该信号并基于信号传递的信息来改变其执行。就这一点而言，信号是一种软件中断。

当输入 Ctrl-C 时，shell 会发送一个 SIGINT 信号到进程。

以下的 Python 程序展示了捕获信号 SIGINT 并忽略它的基本操作，它并不会让程序停止。为了停止这个程序，需要使用 SIGQUIT 信号，通过输入 Ctrl- 可以发送该信号。

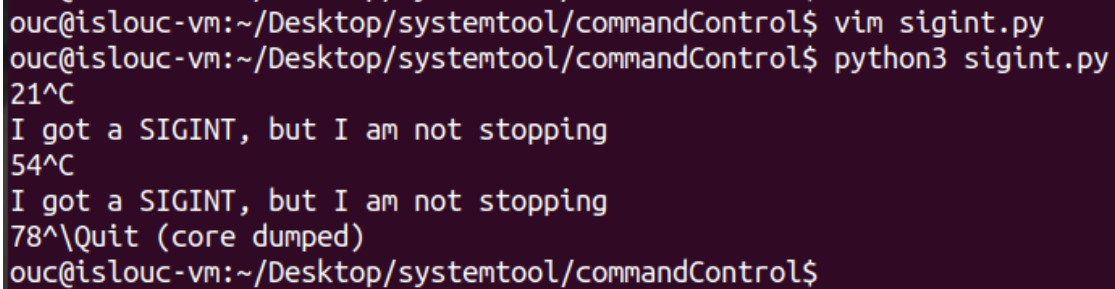
```
#!/usr/bin/env python
import signal, time

def handler(signum, time):
    print("\nI got a SIGINT, but I am not stopping")

signal.signal(signal.SIGINT, handler)
i = 0
while True:
    time.sleep(.1)
```

```
print("\r{}".format(i), end="")  
i += 1
```

向这个程序发送两次 SIGINT，然后再发送一次 SIGQUIT

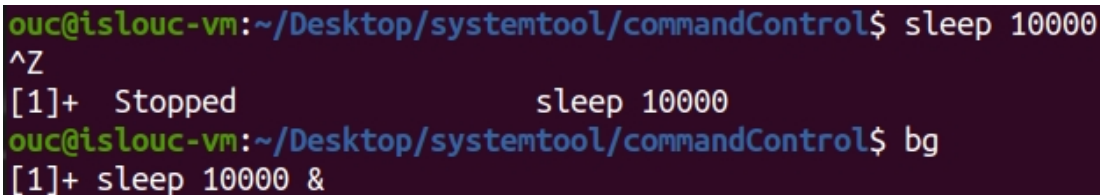


```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim sigint.py  
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 sigint.py  
21^C  
I got a SIGINT, but I am not stopping  
54^C  
I got a SIGINT, but I am not stopping  
78^\Quit (core dumped)  
ouc@islouc-vm:~/Desktop/systemtool/commandControl$
```

Figure 1: 3.1 任务控制

### 3.2 执行 sleep 10000 这个任务。然后用 Ctrl-Z 将其切换到后台并使用 bg 来继续允许它

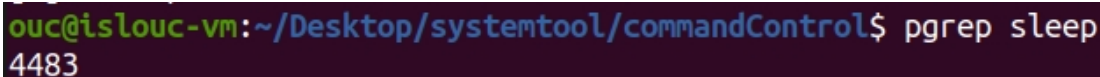
在终端中执行 sleep 10000 这个任务。然后用 Ctrl-Z 将其切换到后台并使用 bg 来继续允许它。



```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ sleep 10000  
^Z  
[1]+  Stopped                  sleep 10000  
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ bg  
[1]+ sleep 10000 &
```

Figure 2: 3.2 sleep 10000

### 3.3 使用 pgrep 来查找 pid



```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ pgrep sleep  
4483
```

Figure 3: 3.3 pgrep 命令

### 3.4 使用 pkill 结束进程

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ pkill -af sleep
pkill: invalid option -- 'a'

Usage:
  pkill [options] <pattern>

Options:
  -<sig>, --signal <sig>    signal to send (either number or name)
  -e, --echo                 display what is killed
  -c, --count                count of matching processes
  -f, --full                 use full process name to match
  -g, --pgroup <PGID,...>   match listed process group IDs
  -G, --group <GID,...>     match real group IDs
  -i, --ignore-case          match case insensitively
  -n, --newest               select most recently started
  -o, --oldest               select least recently started
  -P, --parent <PPID,...>   match only child processes of the given parent
  -s, --session <SID,...>   match session IDs
  -t, --terminal <tty,...>  match by controlling terminal
  -u, --euid <ID,...>        match by effective IDs
  -U, --uid <ID,...>         match by real IDs
  -x, --exact                match exactly with the command name
  -F, --pidfile <file>       read PIDs from file
  -L, --logpidfile           fail if PID file is not locked
  -r, --runstates <state>    match runstates [D,S,Z,...]
  --ns <PID>                 match the processes that belong to the same
                             namespace as <pid>
  --nslist <ns,...>          list which namespaces will be considered for
                             the --ns option.
                             Available namespaces: ipc, mnt, net, pid, user, uts

  -h, --help                 display this help and exit
  -V, --version               output version information and exit

For more details see pgrep(1).
```

Figure 4: 3.4 使用 pkill 结束进程

### 3.5 编写一个 bash 函数 pidwait，它接受一个 pid 作为输入参数，然后一直等待直到该进程结束

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ pidwait()  
> {  
>   while kill -0 $1 #循环直到进程结束  
>   do  
>     sleep 1  
>   done  
>   ls  
> }
```

Figure 5: 3.5 pidwait 函数

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ pidwait $sleep_pid  
[3]+  Done                  sleep 60  
bash: kill: (6524) - No such process  
pidwait.sh  
ouc@islouc-vm:~/Desktop/systemtool/commandControl$
```

Figure 6: 3.5 sleep<sub>pid</sub>

### 3.6 使用 sleep 60 & 作为先执行的程序

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ sleep 60 &  
[3] 6524  
[1] Done                  sleep 60  
[2] Done                  sleep 60
```

Figure 7: 3.6 使用 sleep 60 &

### 3.7 创建一个 dc 别名，它的功能是当我们错误的将 cd 输入为 dc 时也能正确执行

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias dc=cd
```

Figure 8: 3.7 创建一个别名

### 3.8 执行 history 命令来获取您最常用的十条命令，尝试为它们创建别名

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ history | awk '{s1="";print substr($0,2)}' | sort | uniq -c | sort -n | tail -n 10
    5 ssh stu29@10.140.32.159 -p 47029
    6 ./ph 1
    6 ./ph 2
    6 vim
    8 ./dlc bits.c
   10 ./btest
   11 make
   12 ./bomb
   14 gdb bomb
   18 ls
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias ouca='ssh stu29@10.140.32.159 -p 47029'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias oucb='./ph 1'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias oucc='./ph 2'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias oucd='vim'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias ouce='./dlc bits.c'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias oucf='./btest'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias oucg='make'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias ouvh='./bomb'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias ouci='gdb bomb'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ alias oucj='ls'
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ oucj
pidwait.sh
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ █
```

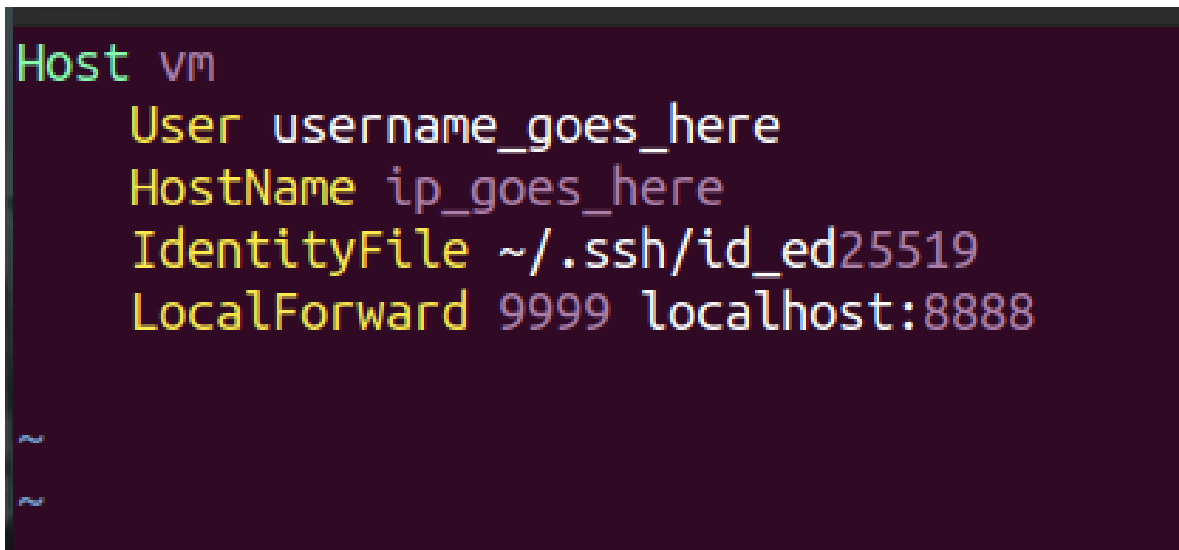
Figure 9: 3.8 为最长用的十条命令创建别名

### 3.9 使用 ssh-keygen -o -a 100 -t ed25519 来创建一个 SSH 密钥对

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ ssh-keygen -o -a 100 -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/ouc/.ssh/id_ed25519): islouc@2023
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in islouc@2023
Your public key has been saved in islouc@2023.pub
The key fingerprint is:
SHA256:PW9kRTXpLLTuemLeEniMTeB6ABKkN7fyufcjYb0SEPs ouc@islouc-vm
The key's randomart image is:
+--[ED25519 256]--+
|    .O.      .OO|
|   .. O   . . O.|
|  . O..+ . . = |
|   . OO.O . + O |
|    . S * X O  |
|   o .E X * o  |
|    o. + = +   |
|    .+ + =..   |
|   .. +.+..+   |
+----[SHA256]-----+
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ ssh-agent
SSH_AUTH_SOCK=/tmp/ssh-0CdWbhYtLtkU/agent.4292; export SSH_AUTH_SOCK;
SSH_AGENT_PID=4293; export SSH_AGENT_PID;
echo Agent pid 4293;
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ ls ~/.ssh
id_ed25519 id_ed25519.pub known_hosts known_hosts.old
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ █
```

Figure 10: 3.9 创建一个 SSH 密钥对

### 3.10 配置 ssh



```
Host vm
    User username_goes_here
    HostName ip_goes_here
    IdentityFile ~/.ssh/id_ed25519
    LocalForward 9999 localhost:8888

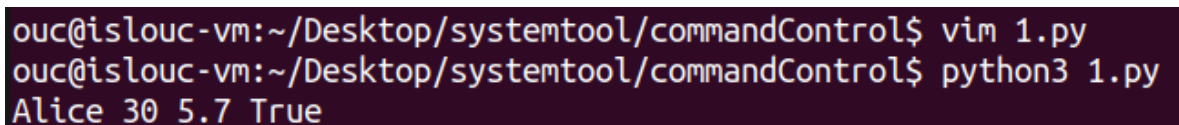
~
~
```

Figure 11: 3.10 配置 SSH

### 3.11 python 语言基础

#### 3.11.1 变量和数据类型

```
name = "Alice" # 字符串
age = 30        # 整数
height = 5.7    # 浮点数
is_student = True # 布尔值
print(name, age, height, is_student)
```



```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 1.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 1.py
Alice 30 5.7 True
```

Figure 12: 3.11.1 变量和数据类型

#### 3.11.2 基本运算

```
x = 10
y = 5
print(x + y) # 加法
print(x - y) # 减法
print(x * y) # 乘法
print(x / y) # 除法
print(x ** y) # 幂运算
```

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 2.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 2.py
15
5
50
2.0
100000
ouc@islouc-vm:~/Desktop/systemtool/commandControl$
```

Figure 13: 3.11.2 基本运算

### 3.11.3 条件语句

```
age = 18
if age >= 18:
    print("Adult")
else:
    print("Not an adult")
```

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 3.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 3.py
Adult
ouc@islouc-vm:~/Desktop/systemtool/commandControl$
```

Figure 14: 3.11.3 条件语句

### 3.11.4 循环语句

```
# for 循环
for i in range(5):
    print(i)

# while 循环
count = 0
while count < 5:
    print(count)
    count += 1
```

```

ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 4.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 4.py
0
1
2
3
4
0
1
2
3
4
ouc@islouc-vm:~/Desktop/systemtool/commandControl$

```

Figure 15: 3.11.4 循环语句

### 3.11.5 定义和调用函数

```

def greet(name):
    return f"Hello, {name}!"

print(greet("Caoyu"))

```

```

ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 5.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 5.py
Hello, Caoyu!
ouc@islouc-vm:~/Desktop/systemtool/commandControl$

```

Figure 16: 3.11.5 定义和调用函数

### 3.11.6 列表

```

fruits = ["apple", "banana", "cherry"]
prin(fruits[0]) # 访问第一个元素
fruits.append("date") # 添加元素
print(fruits)

```



```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 6.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 6.py
apple
['apple', 'banana', 'cherry', 'date']
ouc@islouc-vm:~/Desktop/systemtool/commandControl$
```

Figure 17: 3.11.16 列表

### 3.11.7 读写文件

```
# 写入文件
with open("example.txt", "w") as file:
    file.write("Hello, file!")

# 读取文件
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 7.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 7.py
Hello, file!
```

Figure 18: 3.11.7 读写文件

### 3.11.8 基本异常处理

```
try:
    x = 1 / 0
except ZeroDivisionError:
    print("You can't divide by zero!")
finally:
    print("Execution finished.")
```

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ vim 8.py
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ python3 8.py
You can't divide by zero!
Execution finished.
ouc@islouc-vm:~/Desktop/systemtool/commandControl$
```

Figure 19: 3.11.8 基本异常处理

## 3.12 python 视觉应用

```
ouc@islouc-vm:~/Desktop/systemtool/commandControl$ pip install opencv-python
Collecting opencv-python
  Downloading opencv_python-4.10.0.84-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (62.5 MB)
    | 62.5 MB 21 kB/s
Requirement already satisfied: numpy>=1.17.0; python_version >= "3.7" in /usr/lib/python3/dist-packages (from opencv-python) (1.17.4)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.10.0.84
```

Figure 20: 3.12 配置

### 3.12.1 读取和显示图像

```
import cv2

# 读取图像
image = cv2.imread('bizhi.jpg')

# 显示图像
cv2.imshow('Image', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

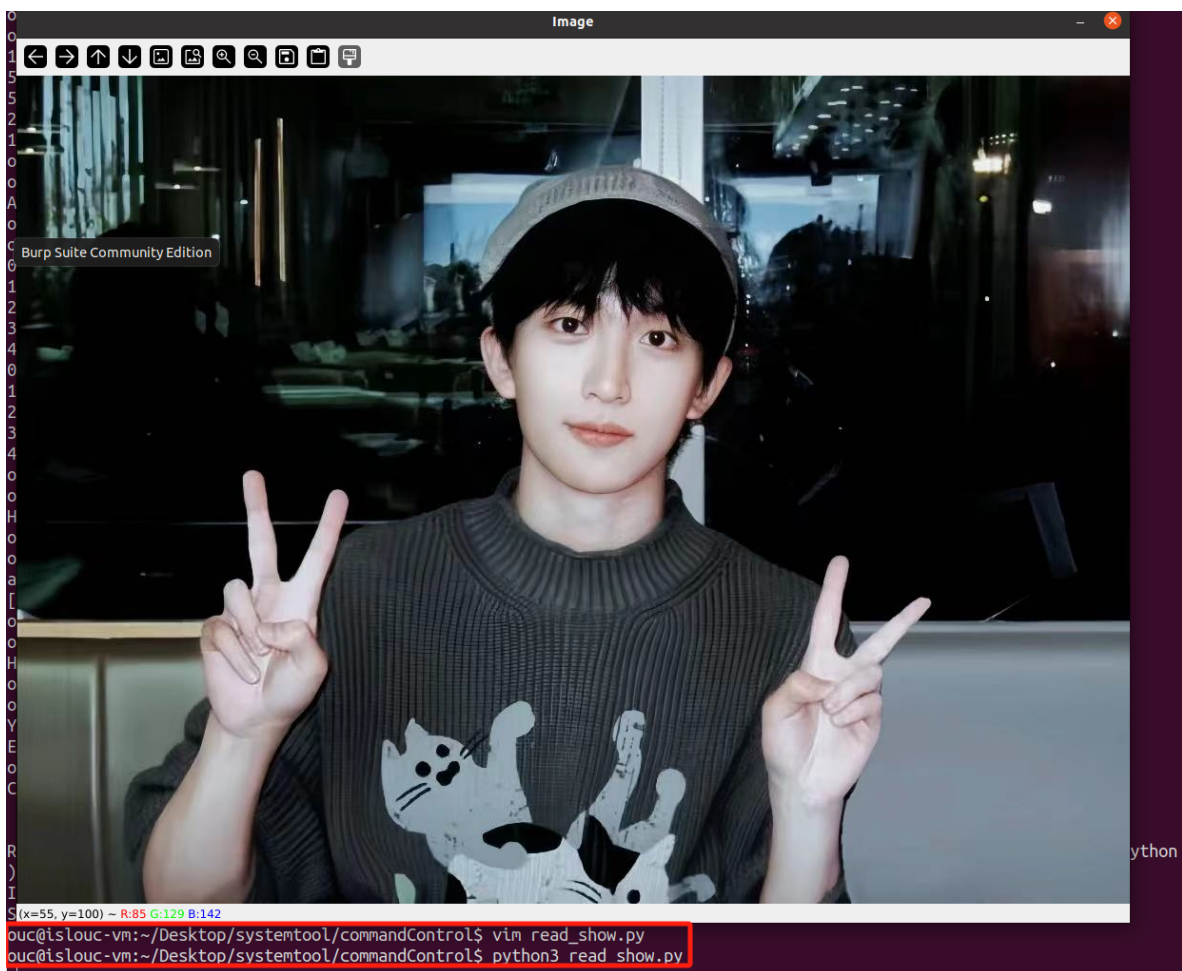


Figure 21: 3.12.1 读取和显示图像

### 3.12.2 图像灰度化

```
import cv2

# 读取图像
image = cv2.imread('bizhi.jpg')

# 转换为灰度图像
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 显示灰度图像
cv2.imshow('Gray Image', gray_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

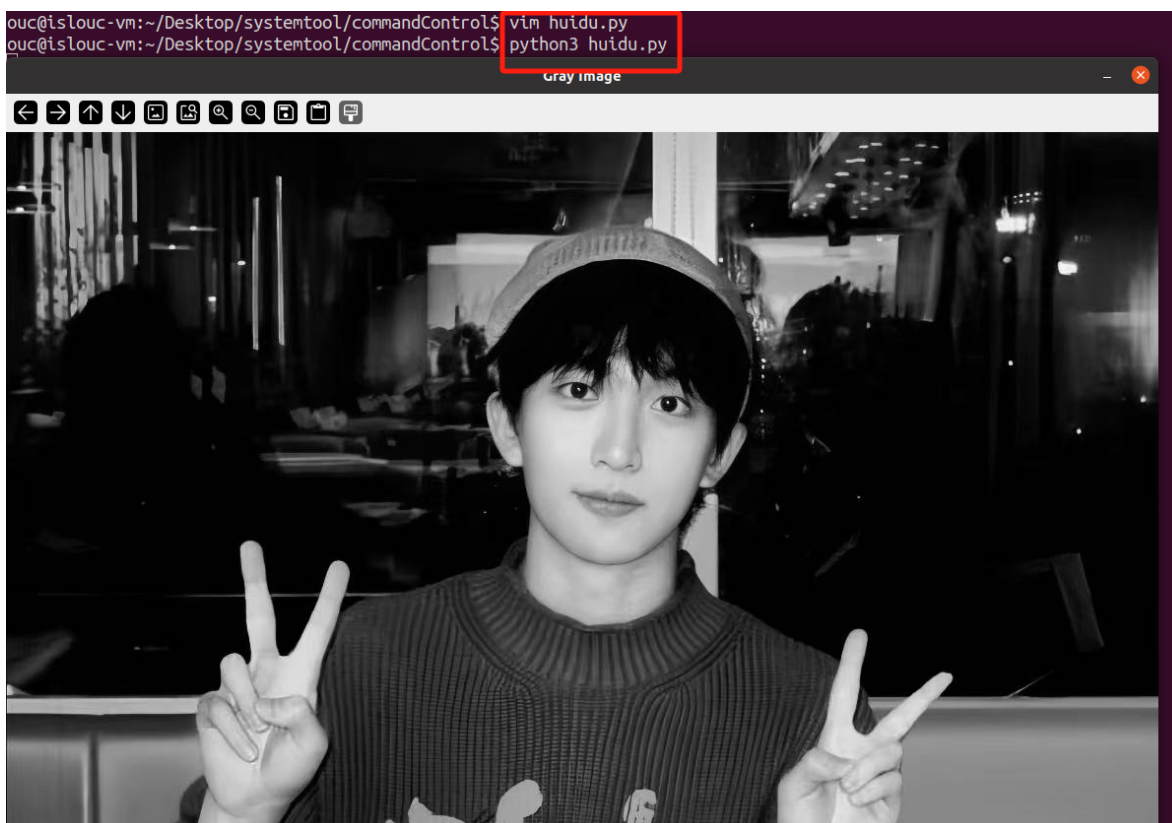


Figure 22: 3.12.2 图像灰度化

### 3.12.3 边缘检测

```
import cv2

# 读取图像
image = cv2.imread('path_to_image.jpg', cv2.IMREAD_GRAYSCALE)

# 边缘检测
edges = cv2.Canny(image, 100, 200)

# 显示边缘图像
cv2.imshow('Edges', edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

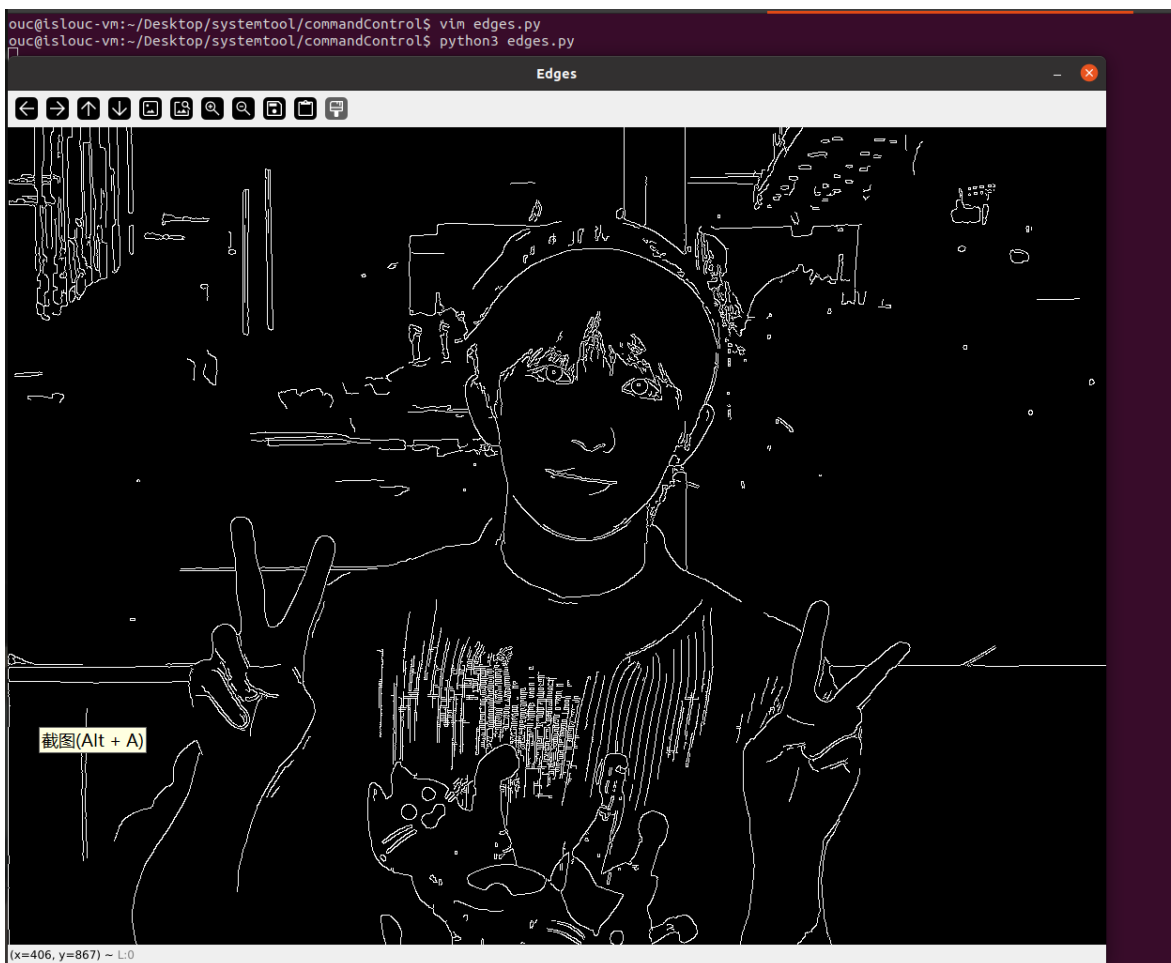


Figure 23: 3.12.3 边缘检测

## 4 实验总结

通过本次实验，我深入了解了命令行环境作为操作系统交互的一种方式。它允许用户通过输入文本命令来执行各种操作，如文件管理和程序运行。

在 Python 开发中，命令行常用于运行脚本、管理虚拟环境以及安装库等。我不仅熟练掌握了命令行环境的相关知识，还深入理解了 Python 基础及其在视觉应用中的使用。通过理论与实践的结合，我对理论知识在实际应用中的实现有了更清晰的理解，特别是在图像处理和计算机视觉领域。

## 5 github 链接

<https://github.com/Caoyu2233/Systemtools.git>