

系统开发工具基础第四周报告

韩昊鑫 24020007039

September 2025

Contents

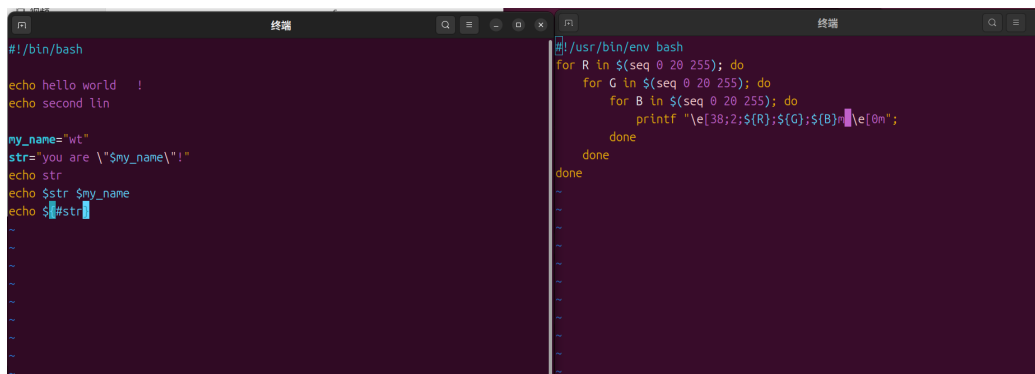
0.1	调试及性能分析	1
0.1.1	可执行文件	1
0.1.2	效果展示 1	1
0.1.3	pdb	2
0.1.4	效果展示 2	2
0.1.5	mypy	2
0.1.6	pdb 用法	2
0.2	元编程	3
0.2.1	生成 PDF01	3
0.2.2	生成 PDF02	3
0.2.3	生成 PDF03	3
0.2.4	PDF 展示	4
0.3	Pytorch	4
0.3.1	张量 1	4
0.3.2	张量 2	4
0.3.3	张量 3	5
0.3.4	梯度	5
0.3.5	神经网络模型	5
0.3.6	损失函数和优化器	6
0.3.7	训练数据	6
0.3.8	测试	6
0.3.9	运行结果	7
0.4	github 链接	8

0.1 调试及性能分析

0.1.1 可执行文件

创建文件，首行指明执行方式，如 `#!/` 告诉系统这个脚本需要什么解释器来执行，即使用哪一种 Shell。

按照 shell 语法可以有如下编写

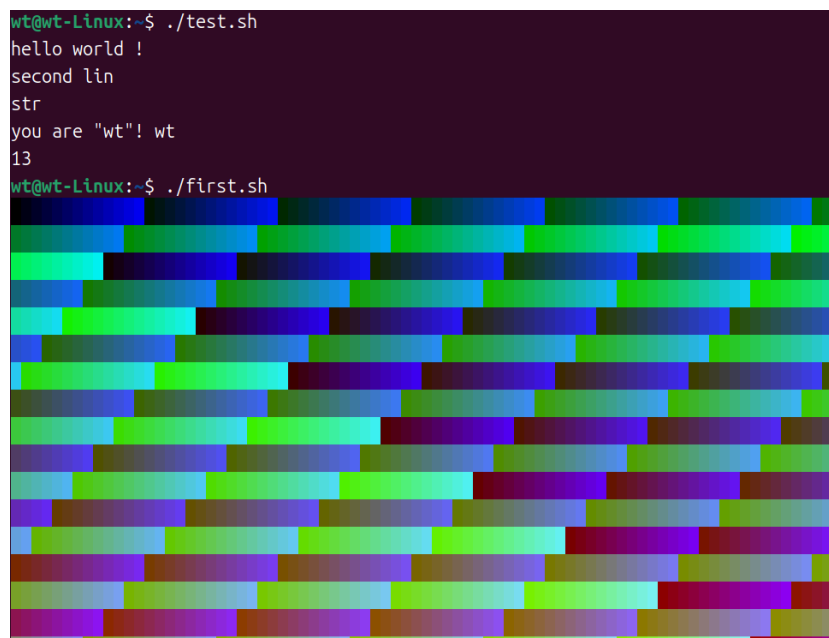


```
#!/bin/bash
echo hello world !
echo second lin

my_name="wt"
str="you are \"${my_name}\"!"
echo str
echo $str $my_name
echo $#str

#!/usr/bin/env bash
for R in $(seq 0 20 255); do
  for G in $(seq 0 20 255); do
    for B in $(seq 0 20 255); do
      printf "\e[38;2;${R};${G};${B}m ";
    done
  done
done
```

0.1.2 效果展示 1



```
wt@wt-Linux:~$ ./test.sh
hello world !
second lin
str
you are "wt"! wt
13
wt@wt-Linux:~$ ./first.sh
```

0.1.3 pdb

在如 Python 文件中 import pdb 可以进行简易调试
它会对 Python 文件进行调试

0.1.4 效果展示 2

```
wt@wt-Linux:~$ vim first.py
wt@wt-Linux:~$ python3 first.py
Traceback (most recent call last):
  File "/home/wt/first.py", line 11, in <module>
    print(bubble_sort([4, 2, 1, 8, 7, 6]))
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "/home/wt/first.py", line 7, in bubble_sort
    if arr[j] > arr[j+1]:
        ~~~^^^^^
IndexError: list index out of range
```

0.1.5 mypy

它会对文件进行静态分析，指出较为明显的错误（编译时错误等）

```
wt@wt-Linux:~$ mypy second.py
second.py:6: error: Incompatible types in assignment (expression has type "int",
variable has type "Callable[[], Any]") [assignment]
second.py:9: error: Incompatible types in assignment (expression has type "float",
variable has type "int") [assignment]
second.py:11: error: Name "baz" is not defined [name-defined]
Found 3 errors in 1 file (checked 1 source file)
```

0.1.6 pdb 用法

1. l(list) - 显示当前行附近的 11 行或继续执行之前的显示；
2. s(step) - 执行当前行，并在第一个可能的地方停止；

3. `n(ext)` - 继续执行直到当前函数的下一条语句或者 `return` 语句;
4. `b(reak)` - 设置断点 (基于传入的参数);
5. `p(rint)` - 在当前上下文对表达式求值并打印结果。还有一个命令是 `pp`, 它使用 `pprint` 打印;
6. `r(eturn)` - 继续执行直到当前函数返回;
7. `q(uit)` - 退出调试器。

0.2 元编程

0.2.1 生成 PDF01

`make` 是最常用的构建系统之一。当您执行 `make` 时, 它会去参考当前目录下名为 `Makefile` 的文件。所有构建目标、相关依赖和规则都需要在该文件中定义, 它看上去是这样的:

```
paper.pdf: paper.tex plot-data.png
———pdflatex paper.tex
plot-%.png: %.dat plot.py
———./plot.py -i $*.dat -o $
```

这个文件中的指令, 即如何使用右侧文件构建左侧文件的规则。或者, 换句话说, 冒号左侧的是构建目标, 冒号右侧的是构建它所需的依赖。缩进的部分是从依赖构建目标时需要用到的一段命令

0.2.2 生成 PDF02

编写驱动文件 `plot.py`, 这是用来告诉系统怎么处理数据的

编写 `latex` 文件 `paper.tex`, 里面存放的是 `latex` 代码, 告诉 `make` 按照什么方式构建

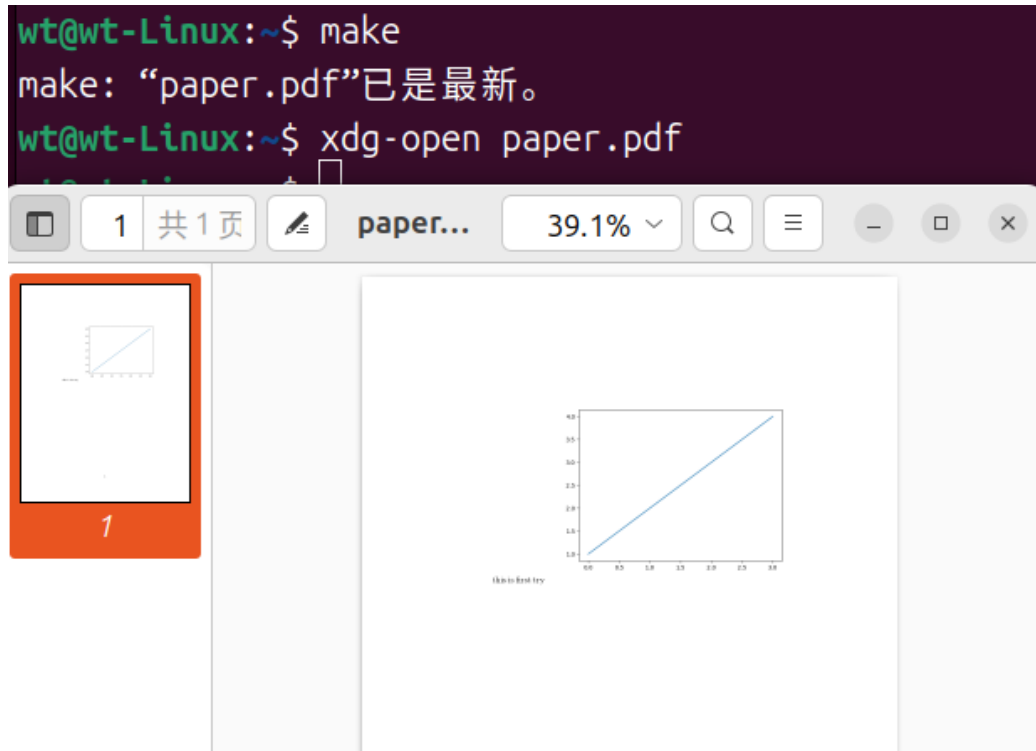
编写数据文件 `data.dat`, 这是用来告诉驱动文件处理哪些数据的

0.2.3 生成 PDF03

安装所有需要的模块, 输入 `make` 即可显示 `paper.pdf` 已更新

找到 `paper.pdf`, 发现生成了一张图片, 这与我们在 `paper.tex` 输入的一致, 同样, 在该文件中输入 `latex` 代码可以得到新的 `pdf`

0.2.4 PDF 展示



0.3 Pytorch

0.3.1 张量 1

```
# 创建一个 2x3 的全 0 张量
a = torch.zeros(2, 3)
# 创建一个 2x3 的全 1 张量
b = torch.ones(2, 3)
# 创建一个 2x3 的随机数张量
c = torch.randn(2, 3)
```

0.3.2 张量 2

```
# 从 NumPy 数组创建张量
import numpy as np
numpy_array = np.array([[1, 2], [3, 4]])
tensor_from_numpy = torch.from_numpy(numpy_array)
```

```

print(tensor_from_numpy)
# 在指定设备（CPU/GPU）上创建张量
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
d = torch.randn(2, 3, device=device)

```

0.3.3 张量 3

```

# 张量相加
e = torch.randn(2, 3)
f = torch.randn(2, 3)
print(e + f)
# 逐元素乘法
print(e * f)
# 张量的转置
g = torch.randn(3, 2)
print(g.t()) # 或者 g.transpose(0, 1)
# 张量的形状
print(g.shape) # 返回形状

```

0.3.4 梯度

```

# 创建一个需要梯度的张量
tensor_requires_grad = torch.tensor([1.0], requires_grad=True)
# 进行一些操作
tensor_result = tensor_requires_grad * 2
# 计算梯度
tensor_result.backward()
print(tensor_requires_grad.grad) # 输出梯度

```

0.3.5 神经网络模型

```

import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
# 定义简单神经网络
class SimpleNN(nn.Module):
    def __init__(self):
        super(SimpleNN, self).__init__()

```

```

self.fc1 = nn.Linear(2, 2)
self.fc2 = nn.Linear(2, 1)
def forward(self, x):
    x = F.relu(self.fc1(x))
    x = self.fc2(x)
    return x
model = SimpleNN()
print(model)

```

0.3.6 损失函数和优化器

```

# 损失函数
criterion = nn.MSELoss()
# 优化器
optimizer = optim.Adam(model.parameters(), lr=0.001)

```

0.3.7 训练数据

```

# 训练数据
X = torch.randn(10, 2)
Y = torch.randn(10, 1)
X_test, Y_test = X, Y
# 训练循环
for epoch in range(100):
    model.train()
    optimizer.zero_grad()
    output = model(X)
    loss = criterion(output, Y)
    loss.backward()
    optimizer.step()
    if (epoch + 1) % 10 == 0:
        print(f'Epoch [epoch+1/100], Loss: {loss.item():.4f}')

```

0.3.8 测试

```

# 测试
model.eval()
with torch.no_grad():

```



```

output = model(X_test)
loss = criterion(output, Y_test)
print(f'Test Loss: {loss.item():.4f}')

```

0.3.9 运行结果

```

de/Code of VScode/pytorch.py"
tensor([[0., 0., 0.],
        [0., 0., 0.]])
tensor([[1., 1., 1.],
        [1., 1., 1.]])
tensor([[ 0.7518,  1.8897,  1.0166],
        [ 0.1225, -0.5498,  2.2059]])
*****
tensor([[1, 2],
        [3, 4]])
tensor([[ 1.1170,  0.3524, -1.5621],
        [ 0.4642, -1.6843, -0.5639]])
*****
tensor([[ 0.4917,  1.9342,  2.4830],
        [ 0.6874, -2.0331,  0.1214]])
tensor([[ -4.0708e-01,  3.1958e-01, -9.5050e-01],
        [ 4.1664e-02,  3.1679e-01,  1.5423e-04]])
tensor([[ 0.5527, -0.1757, -0.6039],
        [ 0.3763, -0.1028, -0.9444]])
torch.Size([3, 2])
*****
tensor([2.])
*****
SimpleNN(
  (fc1): Linear(in_features=2, out_features=2, bias=True)
  (fc2): Linear(in_features=2, out_features=1, bias=True)
)
ReLU:
  tensor([[1.3809, 0.3866],
          [1.2603, 0.7150]])
Sigmoid:
  tensor([[0.7991, 0.5955],
          [0.7791, 0.6715]])
Tanh:
  tensor([[0.8812, 0.3684],
          [0.8512, 0.6138]])
Epoch [10/100], Loss: 1.5204
Epoch [20/100], Loss: 1.4677
Epoch [30/100], Loss: 1.4167
Epoch [40/100], Loss: 1.3677
Epoch [50/100], Loss: 1.3208
Epoch [60/100], Loss: 1.2759
Epoch [70/100], Loss: 1.2333
Epoch [80/100], Loss: 1.1930
Epoch [90/100], Loss: 1.1549
Epoch [100/100], Loss: 1.1192
Test Loss: 1.1157

```

0.4 github 链接

下面是 github 链接:

<https://github.com/C-learning-beginner/git-test.git>