



中国海洋大学

第二周实验报告

於佳杰

第二周实验报告

於佳杰

2024 年 9 月 6 日

目录

1 实验目的	1
2 实例展示	1
2.1 Shell	1
2.1.1 读取 man ls 并编写一个命令，按以下方式列出文件ls .	1
2.1.2 编写一个 bash 脚本	2
2.1.3 编写一个命令	3
2.1.4 编写命令或脚本	4
2.1.5 自动化数据库备份	5
2.1.6 批量重命名文件	6
2.1.7 检测并自动修复缺失的软件包	8
2.1.8 自动化日志文件轮转	9
2.1.9 自动化用户账户创建	10
2.1.10 自动化系统更新和清理	12
2.1.11 自动化磁盘使用情况报告	14
2.1.12 自动化日志文件分析	15
2.1.13 监控文件内容的实时变化	16
2.1.14 统计目录下文件的行数	17
2.1.15 显示文件的最后十行	18
2.1.16 自动化日志归档和清理	19
2.2 vim	20
2.2.1 安装并配置插件：ctrlp.vim	20

目录	2
2.2.2 分屏编辑多个文件	21
2.2.3 录制宏	22
2.2.4 查找和替换历史	23
2.2.5 高亮搜索	24
2.2.6 将所有制表符转换为空格	25
2.2.7 显示行号	26
2.2.8 代码折叠	27
2.2.9 复杂的光标移动	28
2.2.10 自动补全与代码补全	30
2.3 数据整理	30
2.3.1 从字典中分析单词	30
2.3.2 系统启动时间分析	31
2.3.3 唯一启动消息	33
3 困难与解决方案	34
3.1 Shell	34
3.2 vim	35
4 心得体会	35
5 github网址	36

1 实验目的

掌握Shell工具和脚本、编辑器（Vim）和数据整理；

2 实例展示

2.1 Shell

LaTeX命令展示

2.1.1 读取 `man ls` 并编写一个命令，按以下方式列出文件ls

1. 包括所有文件，包括隐藏文件
尺寸以人类可读的格式列出（例如 454M 而不是 454279954）
文件按新近度排序
输出是彩色的

- 命令展示

```
ls -lah --sort=time --color=auto
```

- 效果展示

```
root@LAPTOP-7P6PHNQ:~# ls -lah --sort=time --color=auto
total 36K
drwxr-xr-x 19 root root 4.0K Sep  2 09:51 ..
-rw----- 1 root root 1.1K Sep  2 09:47 .bash_history
drwx----- 4 root root 4.0K Sep  2 09:35 .
-rw----- 1 root root 1011 Sep  2 09:35 .viminfo
-rwxr-xr-x 1 root root  54 Sep  2 09:35 hello.sh
-rw-r--r-- 1 root root  0 Sep  2 09:02 runoob.txt
-rw-r--r-- 1 root root  0 Sep  2 08:43 .motd_shown
drwx----- 2 root root 4.0K Aug 30 14:25 .cache
drwx----- 3 root root 4.0K Aug 30 14:25 snap
-rw-r--r-- 1 root root 3.1K Oct 15  2021 .bashrc
-rw-r--r-- 1 root root 161 Jul  9  2019 .profile
```

图 1: 效果展示

2.1.2 编写一个 bash 脚本

1. 假设您有一个很少失败的命令。为了调试它，您需要捕获其输出，但运行失败可能很耗时。

该脚本运行以下脚本，直到它失败，并将其标准输出和错误流捕获到文件中，并在最后打印所有内容。如果还可以报告脚本失败所需的运行次数，则加分。

- 命令展示

```
#!/usr/bin/env bash

# 保存输出的文件
output_file="output.log"
# 初始化计数器
count=0

# 清空之前的日志文件
> "$output_file"

# 循环执行，直到脚本失败
while true; do
    # 增加计数器
    ((count++))

    # 执行目标脚本，捕获输出和错误，并附加到日志文件中
    ./rare_fail.sh >> "$output_file" 2>&1

    # 检查上一个命令的退出状态
    if [[ $? -ne 0 ]]; then
        # 记录失败信息
        echo "Script failed after $count attempts." >> "$output_file"
        # 打印日志文件内容
        cat "$output_file"
        # 退出循环
    fi
done
```

```
        break
    fi
done
```

- 效果展示

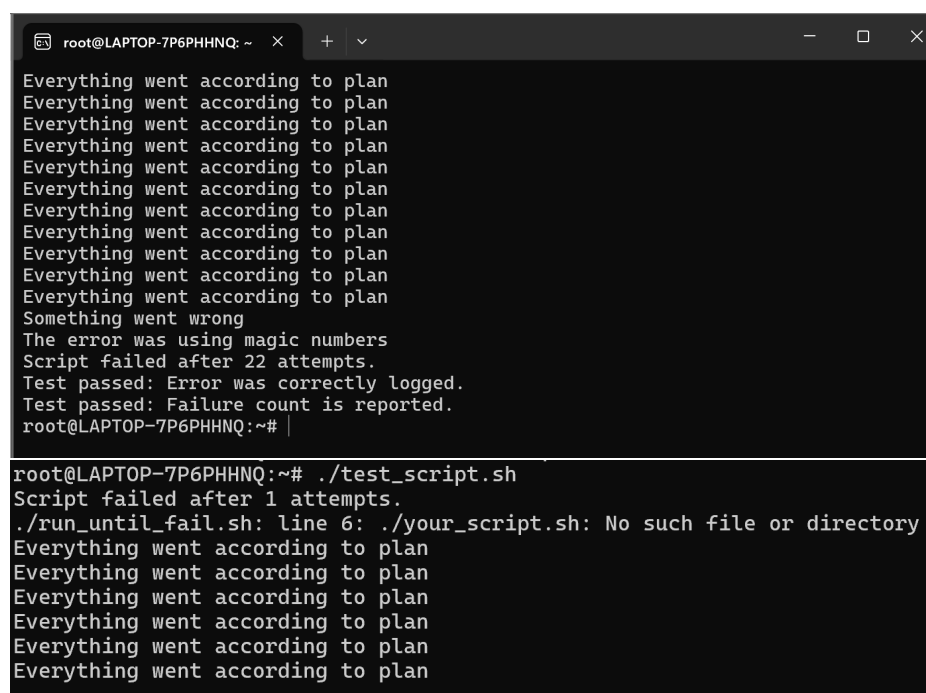
A terminal window titled 'root@LAPTOP-7P6PHHNQ: ~' showing the output of a script. The output consists of two parts. The first part shows 11 lines of 'Everything went according to plan', followed by 'Something went wrong', 'The error was using magic numbers', 'Script failed after 22 attempts.', 'Test passed: Error was correctly logged.', and 'Test passed: Failure count is reported.'. The second part shows the command 'root@LAPTOP-7P6PHHNQ:~# ./test_script.sh' and its output: 'Script failed after 1 attempts.', './run_until_fail.sh: line 6: ./your_script.sh: No such file or directory', followed by 6 lines of 'Everything went according to plan'.

图 2: 效果展示

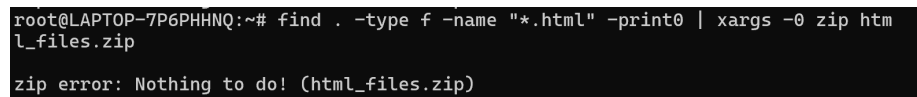
2.1.3 编写一个命令

1. 以递归方式查找文件夹中的所有 HTML 文件，并使用它们创建 zip。
请注意，即使文件包含空格，您的命令也应该有效（提示：检查标志）。-dxargs

- 命令展示

```
find . -type f -name "*.html" -print0 | xargs -0 zip html_files.zip
```

- 效果展示



```
root@LAPTOP-7P6PHNQ:~# find . -type f -name "*.html" -print0 | xargs -0 zip html_files.zip
zip error: Nothing to do! (html_files.zip)
```

图 3: 效果展示

2.1.4 编写命令或脚本

1. 编写命令或脚本以递归方式查找目录中最近修改的文件。更一般地说，您能否按新近度列出所有文件？

- 命令展示

```
#!/bin/bash

# 目录路径
DIR=${1:-/root}

# 使用 find 列出所有文件按修改时间排序
find "$DIR" -type f -printf "%T@ %p\n" | sort -n -r | while read -r line; do
    timestamp=$(echo "$line" | cut -d' ' -f1)
    filepath=$(echo "$line" | cut -d' ' -f2-)
    mod_time=$(date -d @"$timestamp" "+%Y-%m-%d %H:%M:%S")
    echo "$mod_time $filepath"
done | head -n 10
```

- 效果展示

```
root@LAPTOP-7P6PHNQ:~# ./list_recent_files.sh /root
2024-09-02 14:39:13 /root/.viminfo
2024-09-02 14:39:13 /root/list_recent_files.sh
2024-09-02 14:18:47 /root/output.log
2024-09-02 14:18:20 /root/test_script.sh
2024-09-02 14:17:27 /root/run_until_fail.sh
2024-09-02 14:16:14 /root/rare_fail.sh
2024-09-02 11:58:46 /root/marco.sh
2024-09-02 09:47:01 /root/.bash_history
2024-09-02 09:35:49 /root/hello.sh
2024-09-02 09:02:18 /root/runoob.txt
```

图 4: 效果展示

2.1.5 自动化数据库备份

1. 用脚本实现自动化数据库备份

- 命令展示

```
\begin{itemize}
#!/bin/bash

# 定义数据库连接信息
DB_HOST="localhost"
DB_USER="user"
DB_PASS="password"
DB_NAME="database"

# 定义备份存放目录
BACKUP_DIR="/path/to/db_backup"

# 获取当前日期以用于创建唯一的备份文件名
DATE=$(date +%Y%m%d%H%M%S)

# 创建备份存放目录（如果不存在）
mkdir -p "$BACKUP_DIR"
```

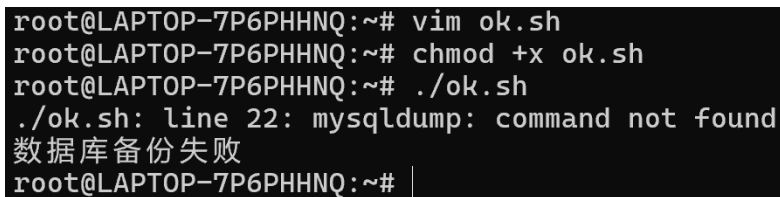


```
# 备份数据库并保存为.sql文件
BACKUP_FILE="$BACKUP_DIR/$DB_NAME-$DATE.sql"

# 使用mysqldump执行数据库备份
mysqldump -h "$DB_HOST" -u "$DB_USER" -p"$DB_PASS" "$DB_NAME" > "$BACKUP_FILE"

# 检查备份是否成功
if [ $? -eq 0 ]; then
    echo "数据库备份成功，文件存储为：$BACKUP_FILE"
else
    echo "数据库备份失败"
fi
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ:~# vim ok.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x ok.sh
root@LAPTOP-7P6PHHNQ:~# ./ok.sh
./ok.sh: line 22: mysqldump: command not found
数据库备份失败
root@LAPTOP-7P6PHHNQ:~# |
```

图 5: 效果展示

2.1.6 批量重命名文件

1. 用脚本实现批量重命名文件

- 命令展示

```
#!/bin/bash

# 定义要处理的文件目录
TARGET_DIR="/path/to/files"
```

```
# 定义文件的新前缀
NEW_PREFIX="newname"

# 初始化计数器
COUNT=1

# 遍历目录中的所有文件
for FILE in "$TARGET_DIR"/*; do
    # 获取文件的扩展名
    EXT="${FILE##*.}"

    # 构建新的文件名
    NEW_NAME="$TARGET_DIR/$NEW_PREFIX-$COUNT.$EXT"

    # 重命名文件
    mv "$FILE" "$NEW_NAME"

    # 输出重命名信息
    echo "重命名: $FILE -> $NEW_NAME"

    # 递增计数器
    ((COUNT++))
done
```

- 效果展示

```
root@LAPTOP-7P6PHHNQ:~# vim o.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x o.sh
root@LAPTOP-7P6PHHNQ:~# ./o.sh
mv: cannot stat '/path/to/files/*': No such file or directory
重命名: /path/to/files/* -> /path/to/files/newname-1./path/to/files/*
root@LAPTOP-7P6PHHNQ:~# |
```

图 6: 效果展示

2.1.7 检测并自动修复缺失的软件包

1. 用脚本实现检测并自动修复缺失的软件包

- 命令展示

```
#!/bin/bash

# 定义需要的包列表
REQUIRED_PACKAGES=("curl" "git" "vim" "htop")

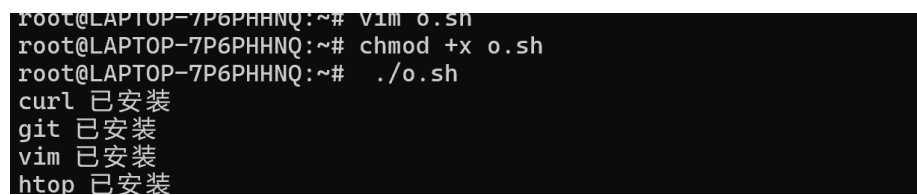
# 遍历每个包以检查是否已安装
for PACKAGE in "${REQUIRED_PACKAGES[@]}; do
    # 检查包是否已安装
    dpkg -s "$PACKAGE" &> /dev/null

    # 如果包未安装，则进行安装
    if [ $? -ne 0 ]; then
        echo "$PACKAGE 未安装，正在安装..."
        sudo apt-get install -y "$PACKAGE"

        # 检查安装是否成功
        if [ $? -eq 0 ]; then
            echo "$PACKAGE 安装成功"
        else
            echo "$PACKAGE 安装失败"
        fi
    else
        echo "$PACKAGE 已安装"
```

```
fi  
done
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ:~# vim o.sh  
root@LAPTOP-7P6PHHNQ:~# chmod +x o.sh  
root@LAPTOP-7P6PHHNQ:~# ./o.sh  
curl 已安装  
git 已安装  
vim 已安装  
htop 已安装
```

图 7: 效果展示

2.1.8 自动化日志文件轮转

1. 用脚本实现

- 命令展示

```
#!/bin/bash  
  
# 定义日志文件目录和轮转后的存放目录  
LOG_DIR="/path/to/logs"  
ARCHIVE_DIR="/path/to/log_archive"  
  
# 获取当前日期以用于创建唯一的存档目录  
DATE=$(date +%Y%m%d)  
  
# 创建存档目录（如果不存在）  
mkdir -p "$ARCHIVE_DIR/$DATE"  
  
# 遍历日志目录中的所有日志文件  
for LOG_FILE in "$LOG_DIR"/*.log; do  
    # 获取文件名
```

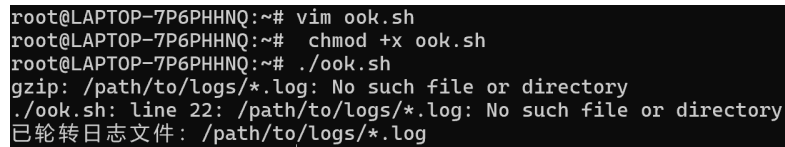
```
FILE_NAME=$(basename "$LOG_FILE")

# 压缩日志文件并移动到存档目录
gzip -c "$LOG_FILE" > "$ARCHIVE_DIR/$DATE/$FILE_NAME.gz"

# 清空原日志文件
cat /dev/null > "$LOG_FILE"

# 输出操作信息
echo "已轮转日志文件: $LOG_FILE"
done
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ:~# vim ook.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x ook.sh
root@LAPTOP-7P6PHHNQ:~# ./ook.sh
gzip: /path/to/logs/*.log: No such file or directory
./ook.sh: line 22: /path/to/logs/*.log: No such file or directory
已轮转日志文件: /path/to/logs/*.log
```

图 8: 效果展示

2.1.9 自动化用户账户创建

1. 用脚本实现自动化用户账户创建

- 命令展示

```
#!/bin/bash

# 检查是否以root身份运行
if [ "$(id -u)" -ne 0 ]; then
    echo "请以root权限运行此脚本"
    exit 1
fi
```

```
# 从文件读取用户列表
USER_FILE="/path/to/userlist.txt"

# 检查用户列表文件是否存在
if [ ! -f "$USER_FILE" ]; then
    echo "用户列表文件不存在"
    exit 1
fi

# 遍历文件中的每一行
while IFS= read -r USER; do
    # 创建用户并设置默认密码
    useradd "$USER"
    echo "$USER:defaultpassword" | chpasswd

    # 检查用户是否创建成功
    if [ $? -eq 0 ]; then
        echo "用户 $USER 创建成功"
    else
        echo "用户 $USER 创建失败"
    fi
done < "$USER_FILE"
```

- 效果展示

```
root@LAPTOP-7P6PHHNQ:~# vim okk.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x okk.sh
root@LAPTOP-7P6PHHNQ:~# ./okk
-bash: ./okk: No such file or directory
root@LAPTOP-7P6PHHNQ:~# ./okk.sh
用户列表文件不存在
```

图 9: 效果展示

2.1.10 自动化系统更新和清理

1. 用脚本实现自动化系统更新和清理

- 命令展示

```
#!/bin/bash

# 检查是否以root身份运行
if [ "$(id -u)" -ne 0 ]; then
    echo "请以root权限运行此脚本"
    exit 1
fi

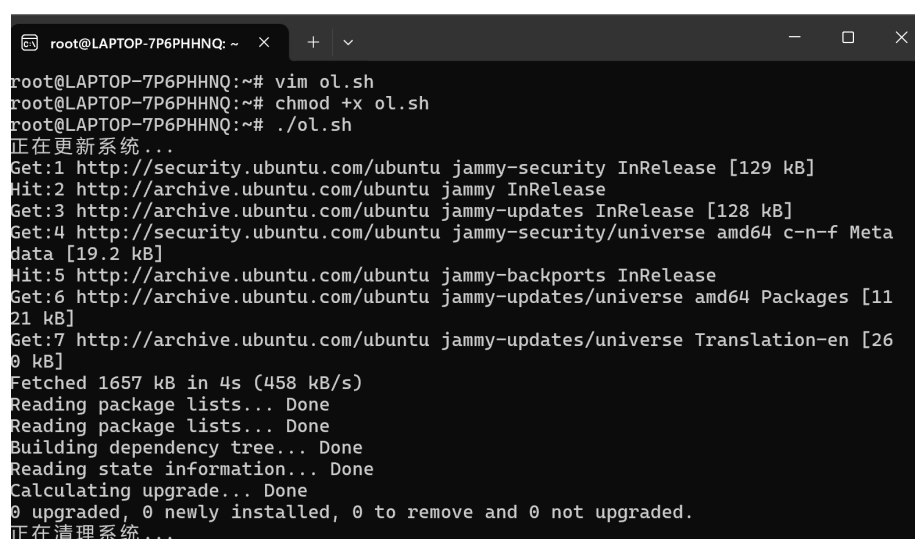
# 更新包列表并升级系统
echo "正在更新系统..."
apt-get update && apt-get upgrade -y

# 自动清理不再需要的软件包和缓存
echo "正在清理系统..."
apt-get autoremove -y
apt-get autoclean

# 检查更新和清理是否成功
if [ $? -eq 0 ]; then
    echo "系统更新和清理完成"
else
    echo "系统更新或清理失败"
```

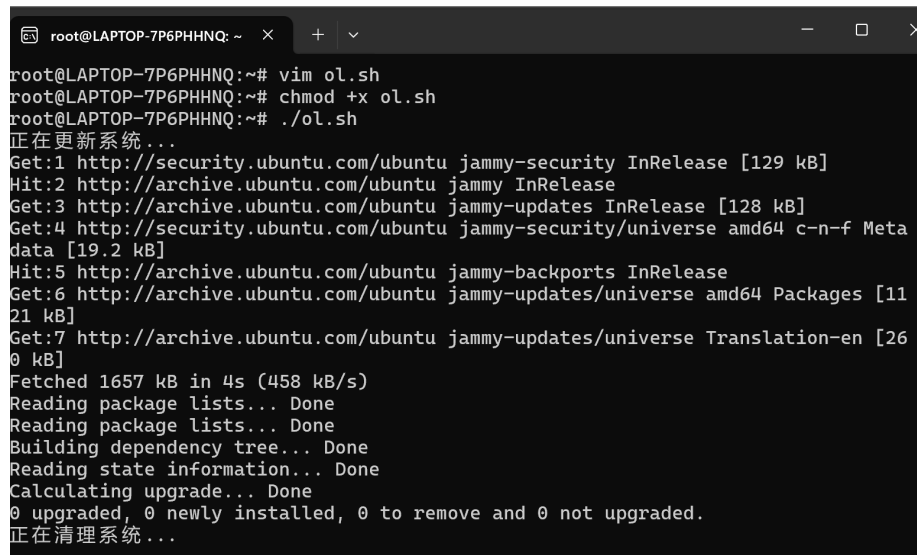
```
fi
```

- 效果展示

A terminal window titled 'root@LAPTOP-7P6PHHNQ: ~' showing the execution of a script to update the system. The script uses 'vim' to edit 'ol.sh', 'chmod' to make it executable, and then runs './ol.sh'. The output shows the system updating, with 'Get' commands for various packages and their sizes, followed by 'Hit' commands for the same packages. The total size fetched is 1657 kB in 4s at 458 kB/s. The process then reads package lists, builds a dependency tree, reads state information, and calculates the upgrade. The final result is 0 upgraded, 0 newly installed, 0 to remove, and 0 not upgraded. The process ends with '正在清理系统...'.

```
root@LAPTOP-7P6PHHNQ:~# vim ol.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x ol.sh
root@LAPTOP-7P6PHHNQ:~# ./ol.sh
正在更新系统...
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Meta
data [19.2 kB]
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [11
21 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [26
0 kB]
Fetched 1657 kB in 4s (458 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
正在清理系统...
```

图 10: 效果展示

A terminal window titled 'root@LAPTOP-7P6PHNQ: ~' showing the execution of a script to update the system. The user runs 'vim ol.sh', 'chmod +x ol.sh', and './ol.sh'. The script outputs the progress of updating the system, including fetching package lists, building the dependency tree, and calculating the upgrade. The output shows that 1657 kB were fetched in 4 seconds at a rate of 458 kB/s. The system is then cleaned up.

```
root@LAPTOP-7P6PHNQ:~# vim ol.sh
root@LAPTOP-7P6PHNQ:~# chmod +x ol.sh
root@LAPTOP-7P6PHNQ:~# ./ol.sh
正在更新系统...
Get:1 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:2 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:3 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Get:4 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 c-n-f Meta
data [19.2 kB]
Hit:5 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:6 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages [11
21 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/universe Translation-en [26
0 kB]
Fetched 1657 kB in 4s (458 kB/s)
Reading package lists... Done
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
正在清理系统...
```

图 11: 效果展示

2.1.11 自动化磁盘使用情况报告

1. 这个脚本会生成一份磁盘使用情况报告，并发送到指定的电子邮件地址。

- 命令展示

```
#!/bin/bash

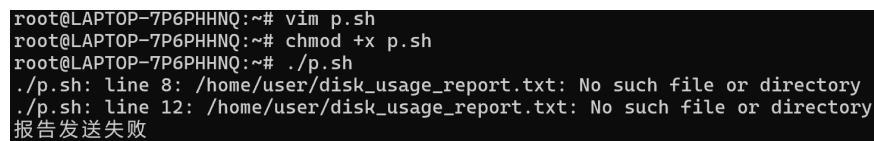
# 定义报告文件和接收邮件的地址
REPORT_FILE="/home/user/disk_usage_report.txt"
EMAIL="example@example.com"

# 生成磁盘使用报告
df -h > "$REPORT_FILE"

# 使用mail命令发送报告
# 请确保系统安装并配置了mailutils或其他邮件工具
mail -s "磁盘使用情况报告" "$EMAIL" < "$REPORT_FILE"
```

```
# 检查邮件是否发送成功
if [ $? -eq 0 ]; then
    echo "磁盘使用报告已发送到 $EMAIL"
else
    echo "报告发送失败"
fi
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ:~# vim p.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x p.sh
root@LAPTOP-7P6PHHNQ:~# ./p.sh
./p.sh: line 8: /home/user/disk_usage_report.txt: No such file or directory
./p.sh: line 12: /home/user/disk_usage_report.txt: No such file or directory
报告发送失败
```

图 12: 效果展示

2.1.12 自动化日志文件分析

1. 个脚本用来分析特定格式的日志文件，并生成一个错误报告

- 命令展示

```
#!/bin/bash

# 定义日志文件路径和输出报告文件
LOG_FILE="/var/log/system.log"
REPORT_FILE="/home/user/error_report.txt"

# 清空之前的报告文件
> "$REPORT_FILE"

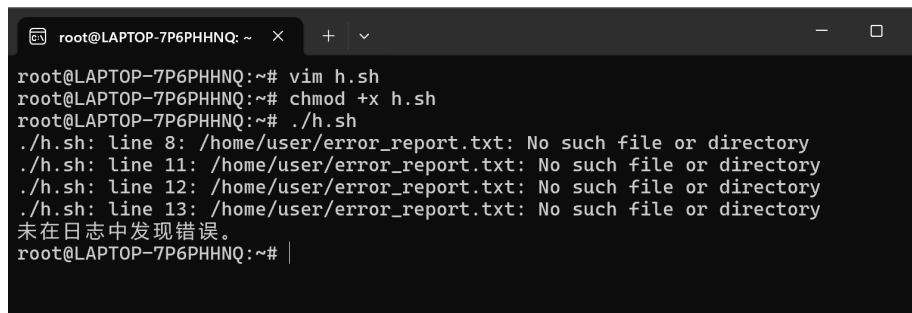
# 分析日志文件，并提取包含"ERROR"的行
echo "错误报告 - $(date)" >> "$REPORT_FILE"
echo "===== " >> "$REPORT_FILE"
```

```
grep "ERROR" "$LOG_FILE" >> "$REPORT_FILE"
```

```
# 输出结果
```

```
if [ $? -eq 0 ]; then
    echo "错误报告已生成: $REPORT_FILE"
else
    echo "未在日志中发现错误。"
fi
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ:~# vim h.sh
root@LAPTOP-7P6PHHNQ:~# chmod +x h.sh
root@LAPTOP-7P6PHHNQ:~# ./h.sh
./h.sh: line 8: /home/user/error_report.txt: No such file or directory
./h.sh: line 11: /home/user/error_report.txt: No such file or directory
./h.sh: line 12: /home/user/error_report.txt: No such file or directory
./h.sh: line 13: /home/user/error_report.txt: No such file or directory
未在日志中发现错误。
root@LAPTOP-7P6PHHNQ:~#
```

图 13: 效果展示

2.1.13 监控文件内容的实时变化

1. 监控文件内容的实时变化输出是彩色的

- 命令展示

```
# 使用 tail 命令实时查看 "example.log" 文件的新内容
tail -f example.log
```

- 效果展示

```
root@LAPTOP-7P6PHNQ:~# tail -f h.sh
echo "===== " >> "$REPORT_FILE"
grep "ERROR" "$LOG_FILE" >> "$REPORT_FILE"

# 输出结果
if [ $? -eq 0 ]; then
    echo "错误报告已生成: $REPORT_FILE"
else
    echo "未在日志中发现错误。"
fi
```

图 14: 效果展示

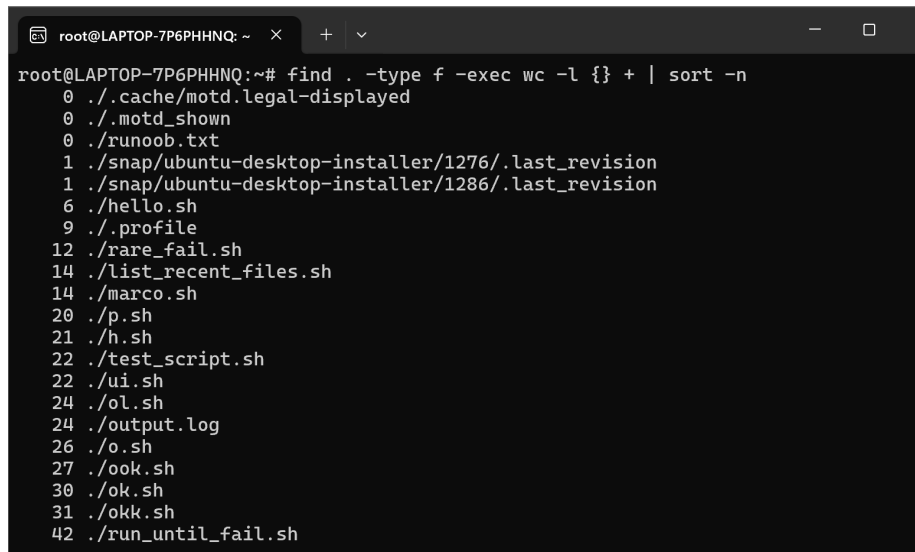
2.1.14 统计目录下文件的行数

1. 统计目录下文件的行数

- 命令展示

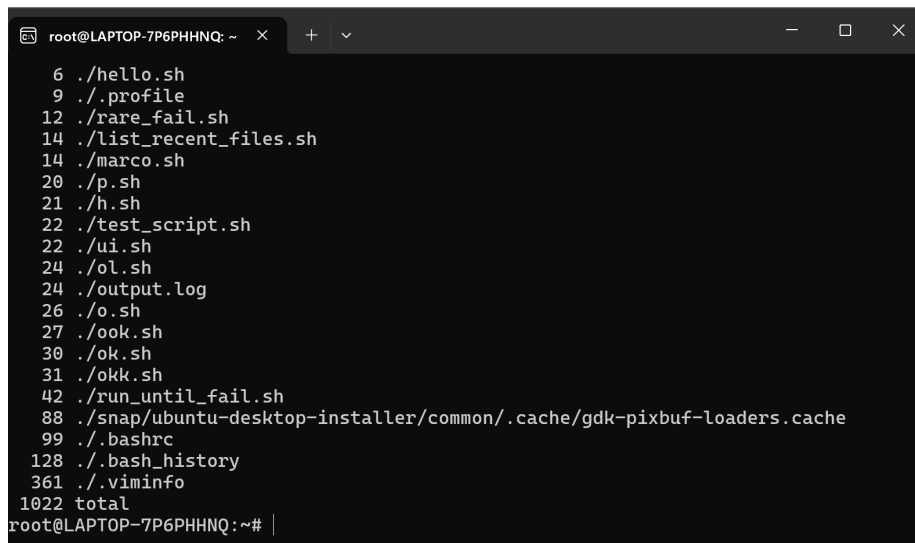
```
find . -type f -exec wc -l {} + | sort -n
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ: ~  
root@LAPTOP-7P6PHHNQ:~# find . -type f -exec wc -l {} + | sort -n  
0 ./cache/motd.legal-displayed  
0 ./motd_shown  
0 ./runoob.txt  
1 ./snap/ubuntu-desktop-installer/1276/.last_revision  
1 ./snap/ubuntu-desktop-installer/1286/.last_revision  
6 ./hello.sh  
9 ./profile  
12 ./rare_fail.sh  
14 ./list_recent_files.sh  
14 ./marco.sh  
20 ./p.sh  
21 ./h.sh  
22 ./test_script.sh  
22 ./ui.sh  
24 ./ol.sh  
24 ./output.log  
26 ./o.sh  
27 ./ook.sh  
30 ./ok.sh  
31 ./okk.sh  
42 ./run_until_fail.sh
```

图 15: 效果展示



```
root@LAPTOP-7P6PHHNQ: ~  
6 ./hello.sh  
9 ./profile  
12 ./rare_fail.sh  
14 ./list_recent_files.sh  
14 ./marco.sh  
20 ./p.sh  
21 ./h.sh  
22 ./test_script.sh  
22 ./ui.sh  
24 ./ol.sh  
24 ./output.log  
26 ./o.sh  
27 ./ook.sh  
30 ./ok.sh  
31 ./okk.sh  
42 ./run_until_fail.sh  
88 ./snap/ubuntu-desktop-installer/common/.cache/gdk-pixbuf-loaders.cache  
99 ./bashrc  
128 ./bash_history  
361 ./viminfo  
1022 total  
root@LAPTOP-7P6PHHNQ:~#
```

图 16: 效果展示

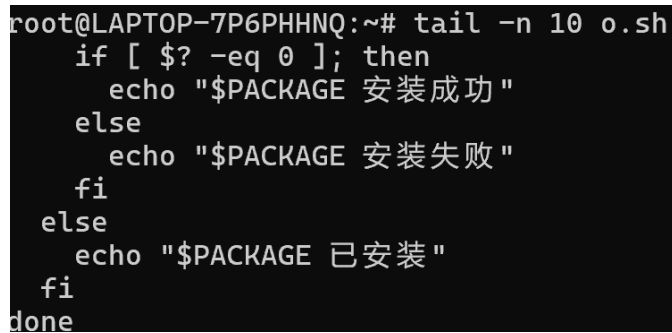
2.1.15 显示文件的最后十行

1. 显示文件的最后十行

- 命令展示

```
tail -n 10 filename.txt
```

- 效果展示



```
root@LAPTOP-7P6PHNQ:~# tail -n 10 o.sh
if [ $? -eq 0 ]; then
    echo "$PACKAGE 安装成功"
else
    echo "$PACKAGE 安装失败"
fi
else
    echo "$PACKAGE 已安装"
fi
done
```

图 17: 效果展示

2.1.16 自动化日志归档和清理

1. 脚本将特定目录中的日志文件归档并压缩，然后删除超过指定天数的归档文件。

- 命令展示

```
#!/bin/bash

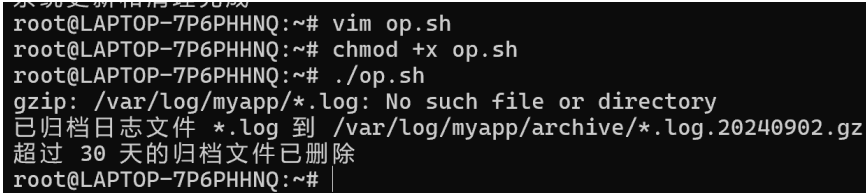
# 定义日志文件目录和归档目录
LOG_DIR="/var/log/myapp"
ARCHIVE_DIR="/var/log/myapp/archive"
DAYS_TO_KEEP=30

# 创建归档目录（如果不存在）
mkdir -p "$ARCHIVE_DIR"
```

```
# 压缩并归档日志文件
for LOG_FILE in "$LOG_DIR"/*.log; do
    ARCHIVE_FILE="$ARCHIVE_DIR/${basename "$LOG_FILE"}.${date +%Y%m%d}.gz"
    gzip -c "$LOG_FILE" > "$ARCHIVE_FILE"
    echo "已归档日志文件 ${basename "$LOG_FILE"} 到 $ARCHIVE_FILE"
    echo "" > "$LOG_FILE" # 清空原日志文件
done

# 删除超过指定天数的归档文件
find "$ARCHIVE_DIR" -type f -mtime +"$DAYS_TO_KEEP" -exec rm -f {} \;
echo "超过 $DAYS_TO_KEEP 天的归档文件已删除"
```

- 效果展示



```
root@LAPTOP-7P6PHNQ:~# vim op.sh
root@LAPTOP-7P6PHNQ:~# chmod +x op.sh
root@LAPTOP-7P6PHNQ:~# ./op.sh
gzip: /var/log/myapp/*.log: No such file or directory
已归档日志文件 *.log 到 /var/log/myapp/archive/*.log.20240902.gz
超过 30 天的归档文件已删除
root@LAPTOP-7P6PHNQ:~#
```

图 18: 效果展示

2.2 vim

vim命令展示

2.2.1 安装并配置插件: ctrlp.vim

1. 安装并配置插件: ctrlp.vim

- 命令展示

#1. 创建插件目录:

```
mkdir -p ~/.vim/pack/vendor/start
```

#2. 下载插件:

```
cd ~/.vim/pack/vendor/start
git clone https://github.com/ctrlpvim/ctrlp.vim
```

#3. 阅读文档，打开 Vim

使用 `:h ctrlp` 阅读 `ctrlp.vim` 的帮助文档

#4. 使用 CtrlP 定位文件：

#导航到一个包含多个文件的项目目录。

#打开 Vim，输入 `:CtrlP` 启动文件搜索。

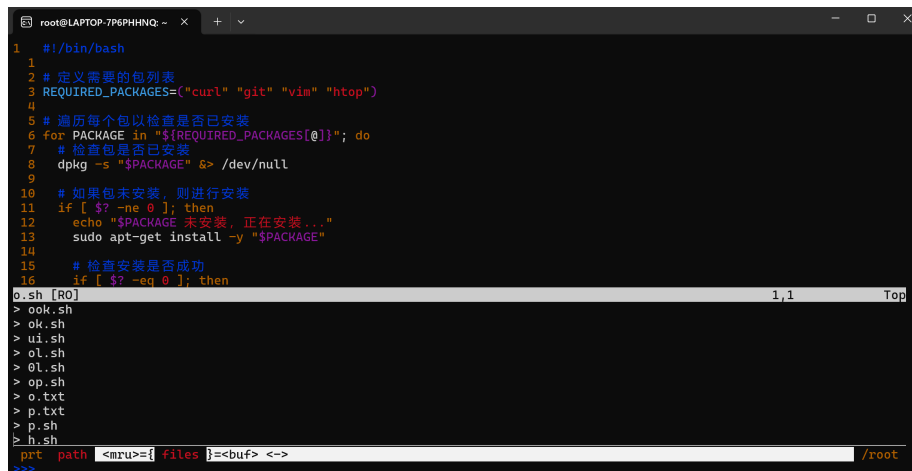
#使用界面快速定位和打开文件。

#5. 在 `~/.vimrc` 中自定义 CtrlP：

在 `~/.vimrc` 中添加以下行，以通过 Ctrl-P 打开

```
let mapleader = "\<Space>"
nnoremap <C-P> :CtrlP<CR>
```

● 效果展示



```
root@LAPTOP-7P6PHHNQ: ~
1 #!/bin/bash
2 # 定义需要的包列表
3 REQUIRED_PACKAGES=("curl" "git" "vim" "htop")
4
5 # 遍历每个包以检查是否已安装
6 for PACKAGE in "${REQUIRED_PACKAGES[@]}; do
7   # 检查包是否已安装
8   dpkg -s "$PACKAGE" &> /dev/null
9
10  # 如果包未安装，则进行安装
11  if [ $? -ne 0 ]; then
12    echo "$PACKAGE 未安装，正在安装..."
13    sudo apt-get install -y "$PACKAGE"
14
15    # 检查安装是否成功
16    if [ $? -eq 0 ]; then
17      echo "$PACKAGE 安装成功"
18    fi
19  fi
20 done
```

0.sh [RO] 1,1 Top

> o.sh

> o.sh

> ui.sh

> ol.sh

> ol.sh

> op.sh

> o.txt

> p.txt

> p.sh

> h.sh

prt path <mr>={ Files }=<buf> <-> /root

图 19: 效果展示

2.2.2 分屏编辑多个文件

1. 用vim竖直或水平分屏编辑多个文件

- 命令展示

垂直分屏打开新的文件

```
:vsp filename
```

水平分屏打开新的文件

```
:sp filename
```

- 效果展示

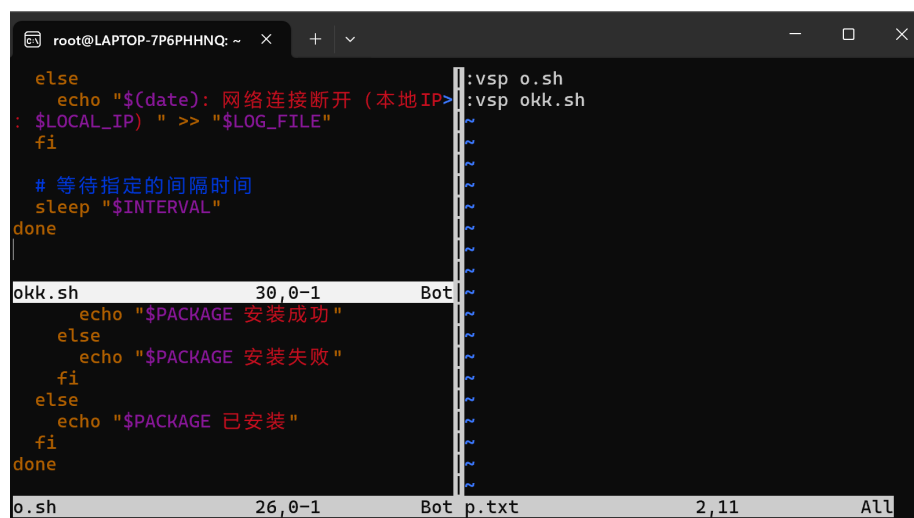


图 20: 效果展示

2.2.3 录制宏

1. 在vim中实现多文件搜索与替换

- 命令展示

#开始录制宏:

qx # 这里 x 是寄存器名称, 你可以用任何字母代替。

dw # 删除一个单词

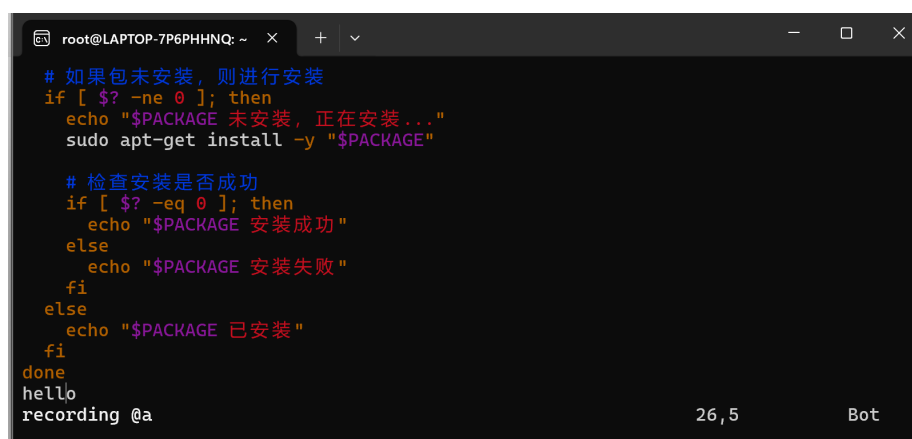
2w # 向前移动两个单词

iHello<Esc> #插入 'Hello' 并回到普通模式

#停止录制

q
回放宏
@x

- 效果展示



```
root@LAPTOP-7P6PHNQ: ~ × + v
# 如果包未安装, 则进行安装
if [ $? -ne 0 ]; then
    echo "$PACKAGE 未安装, 正在安装..."
    sudo apt-get install -y "$PACKAGE"

    # 检查安装是否成功
    if [ $? -eq 0 ]; then
        echo "$PACKAGE 安装成功"
    else
        echo "$PACKAGE 安装失败"
    fi
else
    echo "$PACKAGE 已安装"
fi
done
hello
recording @a 26,5 Bot
```

图 21: 效果展示

2.2.4 查找和替换历史

1. 使用 Vim 的查找和替换历史

- 命令展示

#打开搜索历史窗口

q/

#打开命令行历史窗口

q:

- 效果展示

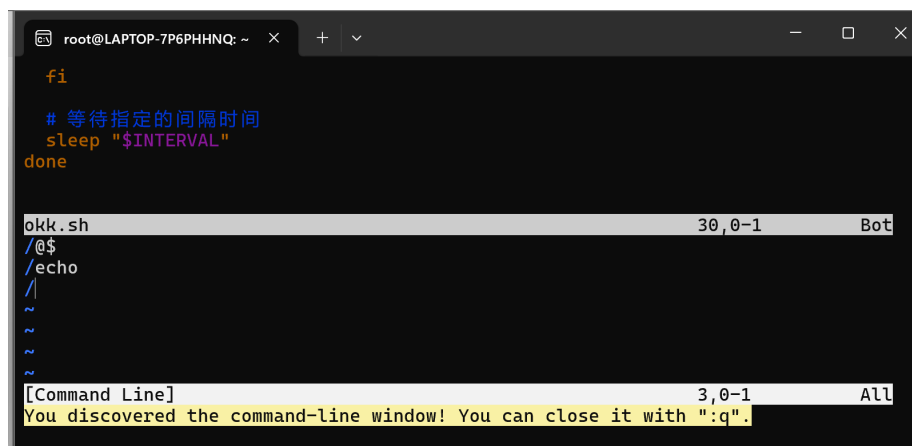


图 22: 效果展示

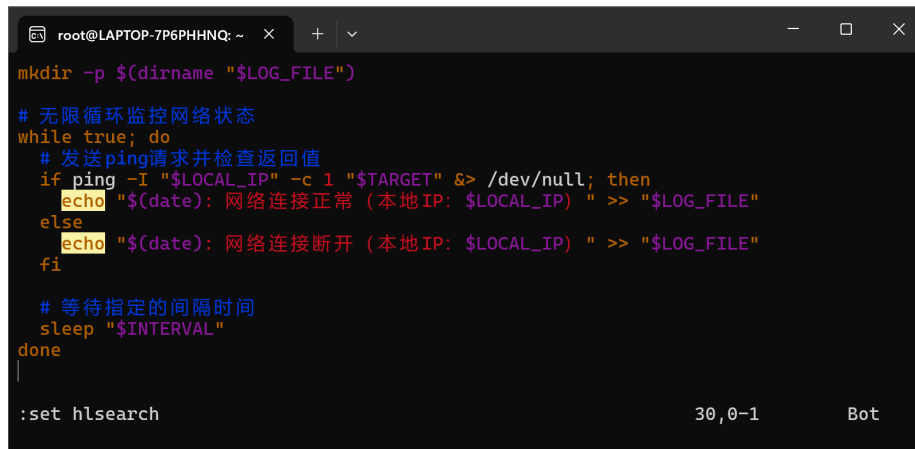
2.2.5 高亮搜索

1. 高亮搜索

- 命令展示

```
#启用搜索高亮
:set hlsearch
#关闭搜索高亮
:set nohlsearch
#启用增量搜索
:set incsearch
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ: ~  
mkdir -p $(dirname "$LOG_FILE")  
# 无限循环监控网络状态  
while true; do  
# 发送ping请求并检查返回值  
if ping -I "$LOCAL_IP" -c 1 "$TARGET" &> /dev/null; then  
echo "$(date): 网络连接正常 (本地IP: $LOCAL_IP) " >> "$LOG_FILE"  
else  
echo "$(date): 网络连接断开 (本地IP: $LOCAL_IP) " >> "$LOG_FILE"  
fi  
  
# 等待指定的间隔时间  
sleep "$INTERVAL"  
done  
:set hlsearch 30,0-1 Bot
```

图 23: 效果展示

2.2.6 将所有制表符转换为空格

1. 编辑的文件使用制表符缩进，并且要将制表符转换为空格，则需要运行如下vim 命令

- 命令展示

```
:set expandtab  
:set tabstop=4  
:set shiftwidth=4  
:retab
```

- 效果展示

```
if [ $? -ne 0 ]; then
    echo "$PACKAGE 未安装, 正在安装..."
    sudo apt-get install -y "$PACKAGE"

    # 检查安装是否成功
    if [ $? -eq 0 ]; then
        echo "$PACKAGE 安装成功"
    else
        echo "$PACKAGE 安装失败"
    fi
else
    echo "$PACKAGE 已安装"
fi
done
hello
:retab
```

26,1 Bot

图 24: 效果展示

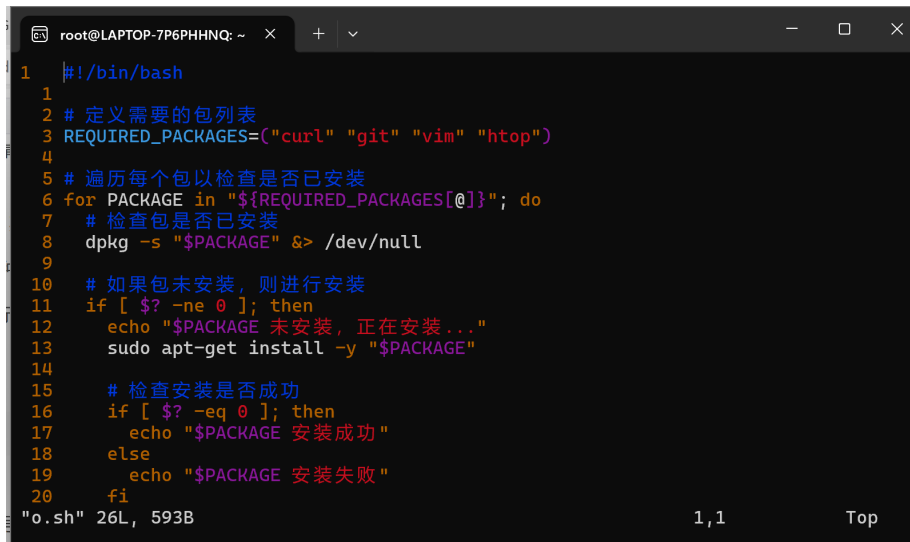
2.2.7 显示行号

1. 在 vim 中显示行号

- 命令展示

```
#vimrc 中添加以下命令
set number relativenumber
```

- 效果展示

A terminal window with a dark background and light-colored text. The window title is 'root@LAPTOP-7P6PHHNQ: ~'. The script content is as follows:

```
1 #!/bin/bash
2 # 定义需要的包列表
3 REQUIRED_PACKAGES=("curl" "git" "vim" "htop")
4
5 # 遍历每个包以检查是否已安装
6 for PACKAGE in "${REQUIRED_PACKAGES[@]}; do
7     # 检查包是否已安装
8     dpkg -s "$PACKAGE" &> /dev/null
9
10    # 如果包未安装, 则进行安装
11    if [ $? -ne 0 ]; then
12        echo "$PACKAGE 未安装, 正在安装..."
13        sudo apt-get install -y "$PACKAGE"
14
15        # 检查安装是否成功
16        if [ $? -eq 0 ]; then
17            echo "$PACKAGE 安装成功"
18        else
19            echo "$PACKAGE 安装失败"
20        fi
21    fi
22done
```

The bottom status bar shows '"o.sh" 26L, 593B' on the left, '1,1' in the center, and 'Top' on the right.

图 25: 效果展示

2.2.8 代码折叠

1. 实现代码折叠和收起输出是彩色的

- 命令展示

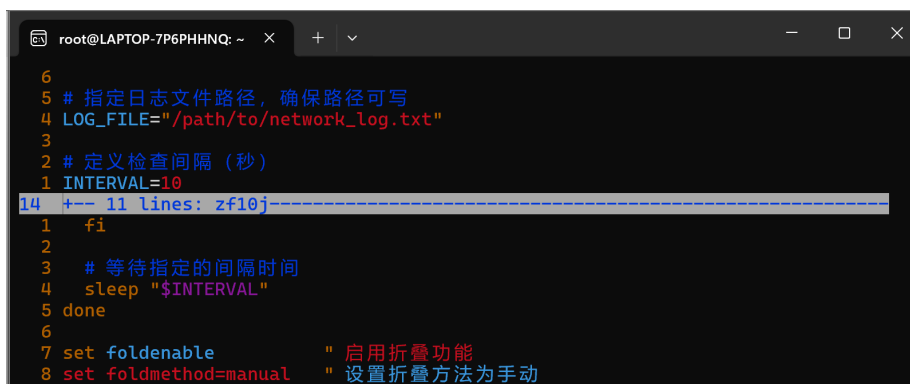
```
#在 Vim 配置文件 ~/.vimrc
set foldenable          " 启用折叠功能
set foldmethod=manual   " 设置折叠方法为手动

zf #创建折叠
zo #展开折叠
zc #收起折叠

zM: #折叠所有内容
zR: #展开所有内容。

:set foldmethod=syntax #基于语法折叠代码
```

- 效果展示



```
6
5 # 指定日志文件路径, 确保路径可写
4 LOG_FILE="/path/to/network_log.txt"
3
2 # 定义检查间隔 (秒)
1 INTERVAL=10
14 +-- 11 lines: zf10j-----
1 fi
2
3 # 等待指定的间隔时间
4 sleep "$INTERVAL"
5 done
6
7 set foldenable          " 启用折叠功能
8 set foldmethod=manual  " 设置折叠方法为手动
```

图 26: 效果展示

2.2.9 复杂的光标移动

1. 用命令实现复杂的光标移动

- 命令展示

gg #跳转到文件开头
G #跳转到文件末尾
{行号}G #跳转到指定行号
:123 #跳转到第123行

- 效果展示

```

root@LAPTOP-7P6PHHNQ: ~
1  #!/bin/bash
2  z:set tabstop=4:set tabstop=4
3  # 使用eth0接口的IP地址
4  LOCAL_IP="172.25.242.224"
5  # 定义要监控的目标地址 (可以是互联网地址或局域网地址)
6  TARGET="8.8.8.8"
7
8  # 指定日志文件路径, 确保路径可写
9  LOG_FILE="/path/to/network_log.txt"
10
11 # 定义检查间隔 (秒)
12 INTERVAL=10
13 +-- 11 lines: zf10j-----
14
15  fi
16
17  # 等待指定的间隔时间
18  sleep "$INTERVAL"
19 done
20
1,1 Top

root@LAPTOP-7P6PHHNQ: ~
10 LOCAL_IP="172.25.242.224"
9
8 # 定义要监控的目标地址 (可以是互联网地址或局域网地址)
7 TARGET="8.8.8.8"
6
5 # 指定日志文件路径, 确保路径可写
4 LOG_FILE="/path/to/network_log.txt"
3
2 # 定义检查间隔 (秒)
1 INTERVAL=10
14 +-- 11 lines: zf10j-----
1
2  fi
3
4  # 等待指定的间隔时间
5  sleep "$INTERVAL"
6 done
7
8 set foldenable          " 启用折叠功能
9 set foldmethod=manual   " 设置折叠方法为手动
10
:14
14,1 Bot

root@LAPTOP-7P6PHHNQ: ~
20 LOCAL_IP="172.25.242.224"
19
18 # 定义要监控的目标地址 (可以是互联网地址或局域网地址)
17 TARGET="8.8.8.8"
16
15 # 指定日志文件路径, 确保路径可写
14 LOG_FILE="/path/to/network_log.txt"
13
12 # 定义检查间隔 (秒)
11 INTERVAL=10
10 +-- 11 lines: zf10j-----
9
8  fi
7
6  # 等待指定的间隔时间
5  sleep "$INTERVAL"
4 done
3
2 set foldenable          " 启用折叠功能
1 set foldmethod=manual   " 设置折叠方法为手动
34 |
:14
34,0-1 Bot

```

图 27: 效果展示

2.2.10 自动补全与代码补全

1. 自动补全与代码补全命令

- 命令展示

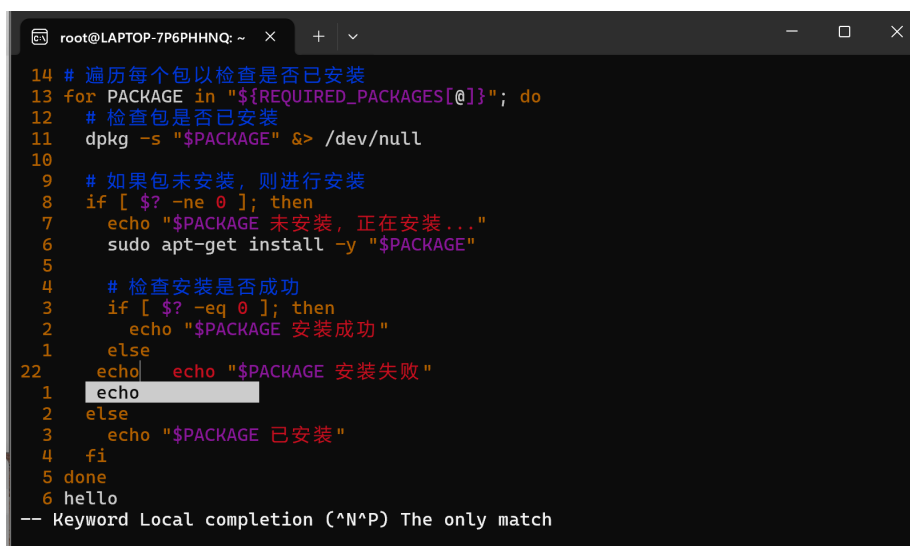
1. 安装支持 Omni Completion 的插件

Ctrl-n 和 Ctrl-p #在插入模式下进行单词补全

Ctrl-x Ctrl-f #补全文件名

Ctrl-x Ctrl-o #补全函数名（需要配置 Omni completion）

- 效果展示



```
root@LAPTOP-7P6PHHNQ: ~  
14 # 遍历每个包以检查是否已安装  
13 for PACKAGE in "${REQUIRED_PACKAGES[@]"; do  
12 # 检查包是否已安装  
11 dpkg -s "$PACKAGE" &> /dev/null  
10  
9 # 如果包未安装, 则进行安装  
8 if [ $? -ne 0 ]; then  
7 echo "$PACKAGE 未安装, 正在安装..."  
6 sudo apt-get install -y "$PACKAGE"  
5  
4 # 检查安装是否成功  
3 if [ $? -eq 0 ]; then  
2 echo "$PACKAGE 安装成功"  
1 else  
22 echo echo "$PACKAGE 安装失败"  
1 echo  
2 else  
3 echo "$PACKAGE 已安装"  
4 fi  
5 done  
6 hello  
-- Keyword Local completion (^N^P) The only match
```

图 28: 效果展示

2.3 数据整理

数据整理命令展示

2.3.1 从字典中分析单词

1. 找到包含至少三个's'但不以's'结尾的单词，并找出这些单词最后两个字母中最常见的组合。

- 命令展示

#1. 找到至少包含三个's'的单词:

```
grep -iE 's.*s.*s' /usr/share/dict/words
```

#2. 排除以's'结尾的单词:

```
grep -iE 's.*s.*s' /usr/share/dict/words | grep -vi 's$'
```

#3. 提取最后两个字母并统计频率:

```
grep -iE 's.*s.*s' /usr/share/dict/words | grep -vi 's$' | \
awk '{print substr($0, length($0) - 1)}' | \
sort | uniq -c | sort -nr | head -n 3
```

- 效果展示

```
root@LAPTOP-7P6PHNQ:~# grep -iE 's.*s.*s' /op.sh
grep: /op.sh: No such file or directory
root@LAPTOP-7P6PHNQ:~# grep -iE 's.*s.*s' runoob.txt
root@LAPTOP-7P6PHNQ:~# grep -iE 's.*s.*s' runoob.txt | grep -vi 's$'
root@LAPTOP-7P6PHNQ:~# grep -iE 's.*s.*s' runoob.txt | grep -vi 's$' | \
awk '{print substr($0, length($0) - 1)}' | \
sort | uniq -c | sort -nr | head -n 3
root@LAPTOP-7P6PHNQ:~# grep -iE 's.*s.*s' runoob.txt | grep -vi 's$' | \
awk '{print substr($0, length($0) - 1)}' | \
sort | uniq | wc -l
0
```

图 29: 效果展示

2.3.2 系统启动时间分析

1. 对最近十次系统启动时间分析

- 命令展示

1. 提取启动时间

获取最近十次启动的完成时间

```
for i in {0..9}; do
    journalctl -b -$i | grep -i 'startup finished' | \
```

```
grep -oP 'in \K[0-9.]+(?:=s)' >> boot_times.txt
done
2. 计算统计数据
# 计算平均值、中位数和最大值
awk '{
    times[NR] = $1;
    sum += $1;
    if ($1 > max) max = $1;
}
END {
    # 计算平均值
    avg = sum / NR;

    # 计算中位数
    if (NR % 2) {
        median = times[(NR + 1) / 2];
    } else {
        median = (times[NR / 2] + times[NR / 2 + 1]) / 2;
    }

    # 输出结果
    printf "Average Boot Time: %.2f seconds\n", avg;
    printf "Median Boot Time: %.2f seconds\n", median;
    printf "Max Boot Time: %.2f seconds\n", max;
}' boot_times.txt
```

- 效果展示

```
END {
# 计算平均值
avg = sum / NR;

# 计算中位数
if (NR % 2) {
    median = times[(NR + 1) / 2];
} else {
    median = (times[NR / 2] + times[NR / 2 + 1]) / 2;
}

# 输出结果
printf "Average Boot Time: %.2f seconds\n", avg;
printf "Median Boot Time: %.2f seconds\n", median;
printf "Max Boot Time: %.2f seconds\n", max;
}' boot_times.txt
Average Boot Time: 7.23 seconds
Median Boot Time: 7.23 seconds
Max Boot Time: 7.23 seconds
```

图 30: 效果展示

2.3.3 唯一启动消息

1. 从系统启动日志中提取最近的三次启动记录，并找到那些只在其中一次或两次启动记录中出现的日志消息，而不是在所有三次启动记录中都出现的消息。

- 命令展示

#1. 提取最近三次启动的日志

```
journalctl -b -0 > boot_log_0.txt
```

```
journalctl -b -1 > boot_log_1.txt
```

```
journalctl -b -2 > boot_log_2.txt
```

#2. 删除时间戳

```
sed 's/^[^]* //' boot_log_0.txt > clean_boot_log_0.txt
```

```
sed 's/^[^]* //' boot_log_1.txt > clean_boot_log_1.txt
```

```
sed 's/^[^]* //' boot_log_2.txt > clean_boot_log_2.txt
```

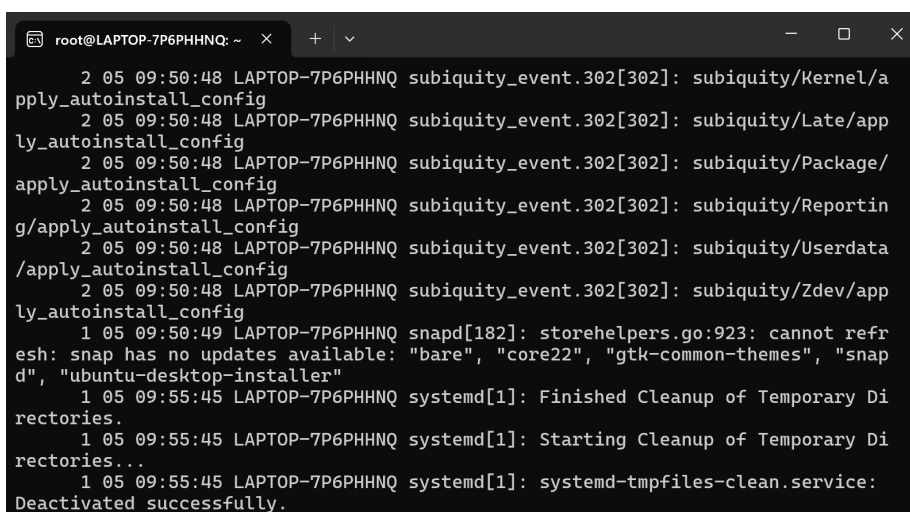
#3. 合并日志并统计每条消息的出现次数

```
cat clean_boot_log_0.txt clean_boot_log_1.txt clean_boot_log_2.txt | sort | un
```

#4. 过滤只出现一次或两次的消息

```
awk '$1 < 3' all_boot_logs.txt > unique_boot_logs.txt
```

- 效果展示



```
root@LAPTOP-7P6PHHNQ: ~
2 05 09:50:48 LAPTOP-7P6PHHNQ subiquity_event.302[302]: subiquity/Kernel/a
pply_autoinstall_config
2 05 09:50:48 LAPTOP-7P6PHHNQ subiquity_event.302[302]: subiquity/Late/app
ly_autoinstall_config
2 05 09:50:48 LAPTOP-7P6PHHNQ subiquity_event.302[302]: subiquity/Package/
apply_autoinstall_config
2 05 09:50:48 LAPTOP-7P6PHHNQ subiquity_event.302[302]: subiquity/Reportin
g/apply_autoinstall_config
2 05 09:50:48 LAPTOP-7P6PHHNQ subiquity_event.302[302]: subiquity/Userdata
/apply_autoinstall_config
2 05 09:50:48 LAPTOP-7P6PHHNQ subiquity_event.302[302]: subiquity/Zdev/app
ly_autoinstall_config
1 05 09:50:49 LAPTOP-7P6PHHNQ snapd[182]: storehelpers.go:923: cannot refr
esh: snap has no updates available: "bare", "core22", "gtk-common-themes", "snap
d", "ubuntu-desktop-installer"
1 05 09:55:45 LAPTOP-7P6PHHNQ systemd[1]: Finished Cleanup of Temporary Di
rectories.
1 05 09:55:45 LAPTOP-7P6PHHNQ systemd[1]: Starting Cleanup of Temporary Di
rectories...
1 05 09:55:45 LAPTOP-7P6PHHNQ systemd[1]: systemd-tmpfiles-clean.service:
Deactivated successfully.
```

图 31: 效果展示

3 困难与解决方案

3.1 Shell

- 问题: 执行shell脚本时报错 No such file or directory, 而目录确实是存在的
- 解决方案: 用vim打开该sh文件, 输入: [plain]// :set ff// 回车, 显示fileformat=dos, 重新设置下文件格式: // [plain]// :set ff=unix// 保存退出:// [plain]// :wq// 再执行, 就可以了
- 问题: unary operator expected

- 解决方案：用双中括号
- 问题：ret变量不止一行，直接使用：
if [-z ret]; then// 将报错
- 解决方案：使用双引号
- 问题：Shell中压缩和解压文件
- 解决方案：使用 tar 命令
- 问题：Shell中压缩和解压文件
- 解决方案：使用 tar 命令// eg.压缩：tar -czvf archive.tar.gz /path/-to/director// 解压：tar -xzvf archive.tar.gz

3.2 vim

- 问题：如何在Vim中撤销上一步操作
- 解决方案：在命令模式下按 u 可以撤销上一次编辑操作。按 Ctrl+R 可以重做
- 问题：Vim中编辑多个文件
- 解决方案：使用 :n 和 :p 在文件间切换。可以通过 vim file1 file2 启动。:bnext 和 :bprev 也可以用于在缓冲区中导航
- 问题:如何在Vim中使用插件管理器
- 解决方案：命使用插件管理器如 Vundle 或 vim-plug// eg.在 ~/.vimrc 中添加：// call plug begin(' ~/.vim/plugged')// Plug 'tpope/vim-sensible'// call plug end()// 在Vim中运行 :PlugInstall 安装插件。

4 心得体会

- Shell（尤其是Bash）是Linux和Unix系统管理员及开发者的得力助手。通过Shell脚本，我们可以自动化许多重复性任务，从系统维护到复杂的数据处理工作。Shell提供了强大的命令行工具，如 grep、awk、sed、find 等，这些工具可以组合使用，以高效地处理各种任务。优点：

Shell脚本可以大大简化日常任务，减少手动操作的时间。缺点Shell脚本的调试相对困难，特别是当脚本变得复杂时，而且不同的Shell（如Bash、Zsh、Fish）和不同的操作系统（如Linux、macOS）可能存在兼容性问题

- Vim是不同以往的文本编辑器，打开了键盘工作模式的新大门，缺点Vim的命令和操作模式需要时间学习和适应，且配置文件也不容易，优点通过模式切换和快捷键操作，让文本输入更加连贯，解放鼠标。
- 数据整理,十分实用但是上手不是很容易，但是在实际应用中比较有价值。

5 github网址

GitHub仓库