Лабораторная работа № 7. Оболочка и скрипты

20 Января 2025 г.

Александр Яннаев, ИУ9-11Б

Цель работы

Получение навыков написания сценариев на «скриптовых» языках.

Скриптовый язык, на котором будет выполняться лабораторная работа, студентом выбирается самостоятельно. Примеры возможных скриптовых языков: JavaScipt (Node.js), Python, Ruby, Lua, Perl, Racket и т.д.

Примем следующие критерии скриптового языка:

- в unix-среде файл можно сделать исполнимым и добавить в начало файла shebang (`#!/путь/до/интерпретатора`),
- в Windows расширение файла *исходного текста* можно добавить в `PATHEXT` и запускать из оболочки `cmd.exe`,
- Microsoft PowerShell (он из соображений безопасности в `PATHEXT` не добавляется, поэтому выписан отдельно).

Задания

При демонстрации результатов работы преподавателю все скрипты должны запускаться командой, содержащей только имя скрипта (т.е. без указания в командной строке пути к скрипту и интерпретатора), то есть так:

```
./myscript arg1 arg2
```

а не так:

```
bash ./myscript.sh arg1 arg2
```

1. На Bash напишите скрипт, который будет запускать долго выполняющуюся программу (напишите скрипт, имитирующий такую программу, скажем, просто ожидающий несколько минут и завершающийся) строго каждые t минут, но так, чтобы одновременно выполнялось не более 1 экземпляра этой программы. Путь к программе и периодичность запуска передавайте в виде аргументов командной строки. Вывод и ошибки запускаемой

программы направляйте в файлы, имена этих файлов формируйте автоматически. Запускаемую программу запрещается убивать.

2. На Bash напишите скрипт, который принимает путь к проекту на языке C и выводит общее число непустых строк во всех файлах `.c` и `.h` указанного проекта. Предусмотрите рекурсивный обход вложенных папок.

3. На выбранном скриптовом языке напишите программу, которая выводит в консоль указанное число строк заданной длины, состоящих из латинских букв, цифр и печатных знаков, присутствующих на клавиатуре. Длину строки и число строк передавайте как аргументы командой строки. Для каких целей можно использовать такую программу? Оформите логику приложения в виде отдельной функции и поместите её в отдельный модуль.

4. На выбранном скриптовом языке напишите аналог утилиты командной строки `nl`, выполняющей нумерацию непустых строк в файле. Программа должна следовать стандартным соглашениям unix-программ.

   Для тестирования этой программы рекомендуется установить `ifconfig` (программа, выводящая статистику сетевых интерфейсов) тестовый вывод для

   ```
   ifconfig | nl
   ```

   и вашей программы должны быть похожими.

Реализация

Файл `1.sh`

```bash
#!/bin/bash

# проверяем количество аргументов
if [ "$#" -ne 2 ]; then
    echo "Использование: $0 <путь_к_программе> <интервал_в_минутах>"
    exit 1
fi

# задаем переменные
PROGRAM_PATH="$1"  # путь к программе
INTERVAL="$2"      # интервал в минутах
PID_FILE="program.pid"  # файл для хранения PID
LOG_DIR="./logs"   # папка для логов

# создаем папку для логов, если её нет
mkdir -p "$LOG_DIR"

# проверяем, запущена ли программа
```

```
is_program_running() {
    if [ -f "$PID_FILE" ]; then
        PID=$(cat "$PID_FILE")
        if ps -p "$PID" > /dev/null; then
            return 0  # процесс работает
        else
            return 1  # процесс завершён
        fi
    fi
    return 1  # если нет файла PID, программа не запущена
}

# функция для запуска программы
run_program() {
    # создаем лог-файл с текущей датой и временем
    LOG_FILE="$LOG_DIR/$(date +'%Y%m%d_%H%M%S').log"
    ERR_LOG_FILE="$LOG_FILE.err"

    # если программа уже запущена, не запускаем её снова
    if is_program_running; then
        echo "Программа уже работает. Пропускаем запуск."
        return
    fi

    # запускаем программу и сохраняем её PID
    echo "Запуск программы: $PROGRAM_PATH"
    "$PROGRAM_PATH" > "$LOG_FILE" 2> "$ERR_LOG_FILE" &
    echo $! > "$PID_FILE"  # сохраняем PID процесса

    echo "Программа запущена с PID $(cat $PID_FILE). Логи пишутся в $LOG_FILE."
}

# основной цикл
while true; do
    run_program
    echo "Ждем $INTERVAL минут перед следующим запуском."
    sleep $(($INTERVAL * 60))  # переводим минуты в секунды
done
```

Файл `long.sh`

```
#!/bin/bash
echo "Программа началась."
sleep 60  # Программа "работает" 1 минуту
echo "Программа завершена!"
```

Файл `2.sh`

```bash
#!/bin/bash

# проверяем, что передан один аргумент
if [[ $# -ne 1 ]]; then
  echo "Использование: $0 путь_к_директории"
  exit 1
fi

project_path=$1

# выводим путь для отладки
echo "Путь, переданный в скрипт: '$project_path'"

# проверяем, существует ли такой путь
if [[ ! -e $project_path ]]; then
  echo "Ошибка: Указанный путь не существует"
  exit 1
fi

# проверяем, является ли путь директорией
if [[ ! -d $project_path ]]; then
  echo "Ошибка: Указанный путь не является директорией"
  exit 1
fi

# инициализируем переменную для подсчёта строк
total_lines=0
echo "Ищем файлы .c и .h в директории: $project_path"

# ищем все файлы с расширением .c и .h
files=$(find "$project_path" -type f \( -name "*.c" -o -name "*.h" \))

# если файлы не найдены, выводим сообщение
if [[ -z "$files" ]]; then
  echo "Нет файлов с расширениями .c и .h в указанной директории."
  exit 1
fi

# обрабатываем каждый найденный файл
for file in $files; do
  echo "Обрабатываем файл: $file"
  # считаем непустые строки
  lines=$(grep -cve '^\s*$' "$file")
  echo "Непустых строк в $file: $lines"
  total_lines=$((total_lines + lines))
done
```

```
# выводим общее количество непустых строк во всех файлах
echo "Общее число непустых строк во всех файлах .c и .h: $total_lines"
```

Файл main.py

```
import sys
from random_string_generator import generate_strings

def main():
    if len(sys.argv) != 3:
        print("Использование: python main.py длина_строки количество_строк")
        sys.exit(1)

    try:
        length = int(sys.argv[1])
        count = int(sys.argv[2])
    except ValueError:
        print("Ошибка: Длина строки и количество строк должны быть числами.")
        sys.exit(1)

    generate_strings(length, count)

if __name__ == "__main__":
    main()
```

Файл random_string_generator.py

```
import random
import string

def generate_random_string(length):
    # символы, которые могут быть использованы в строке
    characters = string.ascii_letters + string.digits + string.punctuation
    return ''.join(random.choice(characters) for i in range(length))

def generate_strings(length, count):
    # генерация count случайных строк длиной length
    for _ in range(count):
        print(generate_random_string(length))
```

Файл usage_of_3rd.txt

```
1.  Тестирование систем безопасности
   Программа может использоваться для проверки систем на уязвимости, такие как:
       Обработка ввода данных (например, SQL-инъекции, XSS-атаки).
     Генерация случайных строк для анализа устойчивости паролей или ключей.

2.  Генерация данных для машинного обучения
```

Для создания датасетов, особенно для задач обработки текста или анализа данных, программа может генерировать искусственные примеры. Это полезно, когда нужно протестировать алгоритмы до получения реальных данных.

3. Обеспечение уникальности данных

Генерация уникальных строк, которые можно использовать как идентификаторы в базах данных, ссылочные ключи, токены для доступа или метки в системах учета.

4. Тестирование производительности программ и баз данных

Нагрузка на базы данных или приложения симулируется большими объемами случайных данных, чтобы оценить их производительность, время отклика и устойчивость.

5. Поддержка инструментов разработки

Программу можно использовать для автоматизированного тестирования или заполнения данных в интерфейсах при разработке:

Автоматизация ввода данных в формы.

Проверка обработки некорректного ввода в системах.

6. Тестирование алгоритмов кодирования и шифрования

Генерация случайных строк полезна для проверки работы алгоритмов, которые обрабатывают текст, включая шифрование, хэширование или кодирование данных.

7. Работа с big data

При моделировании потоков данных или тестировании распределенных систем (например, Kafka или Hadoop) случайные строки могут использоваться как тестовые события.

8. Игровая индустрия

Программа может пригодиться для генерации случайных имён персонажей, кодов доступа или уникальных игровых ключей.

Файл 4.py

```python
#!/usr/bin/env python3

import sys
import argparse

def print_numbered_lines(stream, start_number=1):
    lineno = start_number
    for line in stream:
        if line.strip():  # проверяем, не пустая ли строка
            # выводим номер строки и саму строку
            print(f"{lineno}\t{line}", end='')
            lineno += 1
        else:
            # пустые строки выводим без номера
            print(line, end='')
```

```
    return lineno

def main():
    parser = argparse.ArgumentParser(
        description="Аналог утилиты nl (нумерация непустых строк). "
                    "Если файлов не указано, чтение идёт из стандартного ввода."
    )
    parser.add_argument(
        'files',
        nargs='*',
        help='Файлы для чтения. Если не указаны, чтение идёт из stdin.'
    )
    args = parser.parse_args()

    current_linenumber = 1

    # если файлы не переданы - читаем из stdin
    if not args.files:
        current_linenumber = print_numbered_lines(sys.stdin, current_linenumber)
    else:
        for filename in args.files:
            try:
                with open(filename, 'r') as f:
                    current_linenumber = print_numbered_lines(f, current_linenumber)
            except FileNotFoundError:
                print(f"Ошибка: файл '{filename}' не найден.", file=sys.stderr)
            except PermissionError:
                print(f"Ошибка: нет доступа к файлу '{filename}'.", file=sys.stderr)

if __name__ == '__main__':
    main()
```

Файл `example.txt` и `example2.txt`

```
eno2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 50:eb:f6:8a:ac:18  txqueuelen 1000  (Ethernet)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 16  memory 0x87000000-87020000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Локальная петля (Loopback))
        RX packets 9588  bytes 876801 (876.8 KB)
```

```
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 9588  bytes 876801 (876.8 KB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

wlo1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 10:51:07:17:1c:b3  txqueuelen 1000  (Ethernet)
        RX packets 15119  bytes 9637439 (9.6 MB)
        RX errors 0  dropped 1  overruns 0  frame 0
        TX packets 15171  bytes 2639276 (2.6 MB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Тестирование

```
  ~ cd ~/Рабочий\ стол/lab7
  lab7 chmod +x long.sh
  lab7 chmod +x 1.sh
  lab7 ./1.sh ./long.sh 1
Запуск программы: ./long.sh
Программа запущена с PID 11175. Логи пишутся в ./logs/20250120_171600.log.
Ждем 1 минут перед следующим запуском.
Запуск программы: ./long.sh
Программа запущена с PID 11304. Логи пишутся в ./logs/20250120_171701.log.
Ждем 1 минут перед следующим запуском.
^C
  lab7 chmod +x 2.sh
  lab7 mkdir test_project
  lab7 cd test_project
  test_project echo -e "#include <stdio.h>\n\nint main() {\n    printf(\"Hello, World);\n
return 0;\n}" > file1.c
  test_project echo -e "#ifndef HEADER_FILE\n#define HEADER_FILE\n\nvoid
function();\n\n#endif" > file2.h
  test_project mkdir subdir
  test_project echo -e "#include \"file2.h\"\n\nvoid function() {\n
// Do something\n}" > subdir/file3.c
  test_project cd ~/Рабочий\ стол/lab7
  lab7 ./2.sh ~/test_project
Путь, переданный в скрипт: '/home/keiichi/test_project'
Ищем файлы .c и .h в директории: /home/keiichi/test_project
Обрабатываем файл: /home/keiichi/test_project/file1.c
Непустых строк в /home/keiichi/test_project/file1.c: 5
Обрабатываем файл: /home/keiichi/test_project/subdir/file3.c
Непустых строк в /home/keiichi/test_project/subdir/file3.c: 4
Обрабатываем файл: /home/keiichi/test_project/file2.h
Непустых строк в /home/keiichi/test_project/file2.h: 4
Общее число непустых строк во всех файлах .c и .h: 13
```

```
  lab7 python3 main.py 10 5
\f%pjFuOnA
G,~//)G|D+
%L)Uhb*LFK
SE03wBo|kh
&,%_PYBVQq
  lab7 chmod +x 4.py
  lab7 ./4.py example.txt
1   eno2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
2           ether 50:eb:f6:8a:ac:18  txqueuelen 1000  (Ethernet)
3           RX packets 0  bytes 0 (0.0 B)
4           RX errors 0  dropped 0  overruns 0  frame 0
5           TX packets 0  bytes 0 (0.0 B)
6           TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
7           device interrupt 16  memory 0x87000000-87020000

8   lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
9           inet 127.0.0.1  netmask 255.0.0.0
10          inet6 ::1  prefixlen 128  scopeid 0x10<host>
11          loop  txqueuelen 1000  (Локальная петля (Loopback))
12          RX packets 9588  bytes 876801 (876.8 KB)
13          RX errors 0  dropped 0  overruns 0  frame 0
14          TX packets 9588  bytes 876801 (876.8 KB)
15          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

16  wlo1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
17          ether 10:51:07:17:1c:b3  txqueuelen 1000  (Ethernet)
18          RX packets 15119  bytes 9637439 (9.6 MB)
19          RX errors 0  dropped 1  overruns 0  frame 0
20          TX packets 15171  bytes 2639276 (2.6 MB)
21          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

  lab7 ./4.py example.txt example2.txt

1   eno2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
2           ether 50:eb:f6:8a:ac:18  txqueuelen 1000  (Ethernet)
3           RX packets 0  bytes 0 (0.0 B)
4           RX errors 0  dropped 0  overruns 0  frame 0
5           TX packets 0  bytes 0 (0.0 B)
6           TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
7           device interrupt 16  memory 0x87000000-87020000

8   lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
9           inet 127.0.0.1  netmask 255.0.0.0
10          inet6 ::1  prefixlen 128  scopeid 0x10<host>
11          loop  txqueuelen 1000  (Локальная петля (Loopback))
```

```
12          RX packets 9588  bytes 876801 (876.8 KB)
13          RX errors 0  dropped 0  overruns 0  frame 0
14          TX packets 9588  bytes 876801 (876.8 KB)
15          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

16  wlo1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
17          ether 10:51:07:17:1c:b3  txqueuelen 1000  (Ethernet)
18          RX packets 15119  bytes 9637439 (9.6 MB)
19          RX errors 0  dropped 1  overruns 0  frame 0
20          TX packets 15171  bytes 2639276 (2.6 MB)
21          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

22  eno2: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
23          ether 50:eb:f6:8a:ac:18  txqueuelen 1000  (Ethernet)
24          RX packets 0  bytes 0 (0.0 B)
25          RX errors 0  dropped 0  overruns 0  frame 0
26          TX packets 0  bytes 0 (0.0 B)
27          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
28          device interrupt 16  memory 0x87000000-87020000

29  lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
30          inet 127.0.0.1  netmask 255.0.0.0
31          inet6 ::1  prefixlen 128  scopeid 0x10<host>
32          loop  txqueuelen 1000  (Локальная петля (Loopback))
33          RX packets 9588  bytes 876801 (876.8 KB)
34          RX errors 0  dropped 0  overruns 0  frame 0
35          TX packets 9588  bytes 876801 (876.8 KB)
36          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

37  wlo1: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
38          ether 10:51:07:17:1c:b3  txqueuelen 1000  (Ethernet)
39          RX packets 15119  bytes 9637439 (9.6 MB)
40          RX errors 0  dropped 1  overruns 0  frame 0
41          TX packets 15171  bytes 2639276 (2.6 MB)
42          TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

   lab7 ./4.py
1st string
1   1st string
2nd string
2   2nd string


3rd (4th) string
3   3rd (4th) string
^C
```

Вывод

В ходе выполнения лабораторной работы я получил практический опыт написания скриптов на Bash и Python, что позволило мне углубиться в основы автоматизации и обработки данных. Работа над заданием научила меня нескольким важным аспектам: 1. **Организация и управление процессами:**
В первом задании я реализовал скрипт, который периодически запускает программу, гарантируя, что одновременно выполняется только один её экземпляр. Это дало мне понимание работы с PID-файлами и управлениями процессами в Unix-среде, а также показало, насколько важно логирование для отладки и последующего анализа работы программ.

2. **Рекурсивный поиск и обработка файлов:**
   Второе задание требовало подсчитать количество непустых строк в файлах с расширениями `.c` и `.h` в указанной директории и её подкаталогах. Этот скрипт позволил мне попрактиковаться в использовании утилиты `find` и обработке текстовых файлов с помощью `grep`.

3. **Генерация случайных данных:**
   Я понял, как можно модульно организовать код, разделив логику генерации строк и основную программу. Это умение очень полезно для дальнейшей разработки, так как позволяет создавать более гибкие и переиспользуемые компоненты.

4. **Нумерация строк как аналог утилиты `nl`:**
   Четвёртое задание помогло мне разобраться, как можно обрабатывать ввод из файла или стандартного ввода, а также как оформлять вывод в соответствии с соглашениями Unix-программ.

Для меня эта лабораторная работа стала хорошей возможностью увидеть, как простые скриптовые языки позволяют решать реальные задачи автоматизации и обработки информации.