Программирование в командном процессоре ОС UNIX. Командные файлы

Лабораторная работа №12

Козомазов Владимир Романович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Цель работы “Программирование в командном процессоре ОС UNIX. Командные файлы” заключается в освоении навыков создания и выполнения командных файлов (shell-скриптов) в UNIX-подобных операционных системах.

2 Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя в другую директорию backup в моём домашнем каталоге.
2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории.

3 Теоретическое введение

3.0.0.1 1. Командный процессор (shell) в UNIX

Командный процессор (интерпретатор команд, *shell*) — это программа, обеспечивающая взаимодействие пользователя с операционной системой UNIX. Он принимает команды, выполняет их и возвращает результат.

Популярные shell-процессоры:

- **Bash (Bourne-Again SHell)** — стандартный в большинстве Linux-дистрибутивов.
- **sh (Bourne Shell)** — более старый, но совместимый с Bash.
- **zsh, ksh, csh** — альтернативные оболочки с дополнительными возможностями.

3.0.0.2 2. Командные файлы (shell-скрипты)

Командный файл — это текстовый файл, содержащий последовательность команд для выполнения в shell.

Особенности shell-скриптов:

- Исполняются интерпретатором (не требуют компиляции).
- Могут принимать аргументы командной строки.
- Поддерживают переменные, условия, циклы, функции.
- Могут вызывать другие программы и скрипты.

3.0.0.3 3. Основные элементы shell-программирования

3.0.0.3.1 3.1. Структура скрипта

```
#!/bin/bash  # Шебанг (указывает интерпретатор)
# Комментарии начинаются с #

echo "Hello, World!"  # Простая команда
```

3.0.0.3.2 3.2. Переменные

- Объявление: `VAR=value` (без пробелов!).
- Использование: `$VAR` или `${VAR}`.

```
name="User"
echo "Hello, $name!"  # Hello, User!
```

3.0.0.3.3 3.3. Параметры командной строки

- `$0` — имя скрипта.
- `$1`, `$2`, ... — аргументы.
- `$#` — количество аргументов.
- `$*` или `$@` — все аргументы.

3.0.0.3.4 3.4. Управляющие конструкции

- **Условия:**

```
if [ "$1" -eq 10 ]; then
    echo "Аргумент равен 10"
else
```

```
    echo "Аргумент не равен 10"
fi
```

- **Циклы:**

```
for file in *.txt; do
    echo "Обработка $file"
done
```

3.0.0.3.5 3.5. Перенаправление ввода/вывода

- `>` — вывод в файл (перезапись).
- `>>` — вывод в файл (дополнение).
- `<` — ввод из файла.
- `|` — конвейер (передача вывода одной команды на вход другой).

3.0.0.3.6 3.6. Коды возврата и обработка ошибок

- `$?` — код завершения последней команды (0 — успех, иначе ошибка).
- `set -e` — завершить скрипт при ошибке.
- `trap` — перехват сигналов.

3.0.0.4 4. Пример скрипта

```
#!/bin/bash
# Скрипт для поиска файлов и вывода информации

if [ $# -eq 0 ]; then
```

```
    echo "Использование: $0 <каталог>"
    exit 1
fi

dir=$1
echo "Файлы в каталоге $dir:"
find "$dir" -type f -exec ls -l {} \;
```

3.0.0.5 5. Практическое применение

- Автоматизация рутинных задач (резервное копирование, логирование).
- Обработка текстовых данных (логи, CSV).
- Управление системными процессами.

4 Выполнение лабораторной работы

Установил менеджер паролей pass с помощью команды `sudo dnf install pass pass-otp` (рис. 4.1).

Рис. 4.1: Установка менеджера паролей

Завершил установку менеджера паролей командой `sudo dnf install gopass` (рис. 4.2).

Рис. 4.2: Завершение установки менеджера паролей

Просмотрел список gpg ключей при помощи команды `gpg --list-secret-keys` (рис. 4.3).

Рис. 4.3: Просмотр списка ключей

Инициализировал хранилище, написав команду ‘pass init (рис. 4.4).

Рис. 4.4: Инициализация хранилища

Создал структуру с git командой `pass git init` (рис. 4.5).

Рис. 4.5: Создание структуры

Синхронизировался с git командами `pass git pull`, `pass git push`
Вручную закоммитил и выложил изменения командами (рис. 4.6).

Рис. 4.6: Ручное выкладывание изменений

Проверил статус синхронизации командой `pass git status` (рис. 4.7).

Рис. 4.7: Проверка статуса синхронизаций

Добавил новый пароль командой `pass insert [OPTIONAL DIR]/[FILENAME]` (рис. 4.8).

Рис. 4.8: Добавление нового пароля

Отобразил пароль для указанного имени файла `pass [OPTIONAL DIR]/[FILENAME]`
Установил дополнительное программное обеспечение (рис. 4.9).

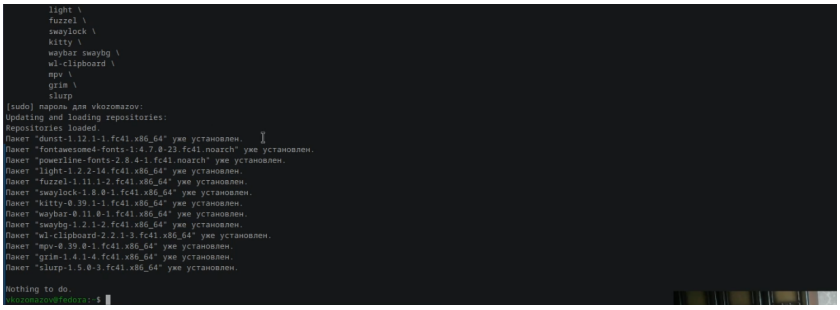

Рис. 4.9: Установка дополнительного программного обеспечения

Установил дополнительно шрифты командами `sudo dnf copr enable peterwu/iosevka`, `sudo dnf search iosevka`, `sudo dnf install iosevka-fonts iosevka-aile-fonts iosevka-curly-fonts iosevka-slab-fonts iosevka-etoile-fonts iosevka-term-fonts` (рис. 4.10).

Рис. 4.10: Установка дополнительных шрифтов

Установил бинарный файл командой `sh -c "$(wget -qO- chezmoi.io/get)"`

Создал собственный репозиторий при помощи утилит на основе шаблона

Подключил репозиторий к своей системе

Использовал chezmoi на нескольких машинах

Настроил новую машину с помощью одной команды

Включил функцию автоматической фиксации и отправлении изменений в репозиторий

5 Выводы

В ходе выполнения работы были изучены ключевые аспекты создания и использования командных файлов (shell-скриптов) в UNIX-подобных системах.

Список литературы