Программирование в командном процессоре ОС UNIX. Расширенное программирование

Лабораторная работа №14

Козомазов Владимир Романович

Содержание

Список иллюстраций

Список таблиц

1 Цель работы

Главная цель данной работы — **углублённое освоение методов и приёмов программирования в командных процессорах UNIX** (Bash, sh, ksh и др.) для создания эффективных, надёжных и безопасных shell-скриптов, способных решать сложные задачи системного администрирования, автоматизации и обработки данных.

2 Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени t2<>t1, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом)
2. Реализовать команду man с помощью командного файла. Изучите содержимое каталога /usr/share/man/man1.
3. Используя встроенную переменную $RANDOM, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита.

3 Теоретическое введение

3.0.0.1 1. Понятие командного процессора в UNIX

Командный процессор (shell) — это интерпретатор команд, обеспечивающий взаимодействие пользователя с операционной системой. В UNIX-подобных системах распространены: - **Bash** (Bourne-Again SHell) — наиболее популярный, с расширенными возможностями - **sh** (Bourne Shell) — стандартный, но с ограниченным функционалом - **zsh/ksh** — альтернативные оболочки с дополнительными функциями

3.0.0.2 2. Особенности shell-программирования

1. **Интерпретируемый характер**:
    - Скрипты выполняются построчно без предварительной компиляции
    - Требуют указания интерпретатора в первой строке (`#!/bin/bash`)
2. **Работа с процессами**:
    - Каждая команда запускает новый процесс
    - Механизмы управления: `&`, `jobs`, `fg`, `bg`, `kill`
3. **Перенаправление ввода-вывода**:
    - Основные операторы: >, <, >>, |, 2>
    - Особые файлы: `/dev/null`, `/dev/zero`

3.0.0.3 3. Уровни программирования в shell

Уровень	Характеристики	Примеры
Базовый	Простые линейные скрипты	Автоматизация рутинных команд
Продвинутый	Использование условий, циклов, функций	Системный мониторинг, обработка данных
Профессиональный	Интеграция с другими языками, сложная логика	DevOps-инструменты, CI/CD системы

3.0.0.4 4. Ключевые технологии расширенного программирования

1. **Регулярные выражения**:
   - Реализации в `grep`, `sed`, `awk`
   - Пример: `grep -E "^[A-Za-z0-9_]+@[a-z]+\.[a-z]{2,4}$" emails.txt`
2. **Работа с текстом**:
   - **awk** — язык обработки строк с возможностью программирования

```
awk '{print $1, $3}' access.log | sort | uniq -c
```

3. **Параллельное выполнение**:
   - Механизмы: `&`, `wait`, `xargs -P`, `GNU parallel`

```
find . -type f -print0 | xargs -0 -P 4 md5sum
```

3.0.0.5 5. Безопасность shell-скриптов

1. **Типичные уязвимости**:
   - Инъекции через непроверенные параметры
   - Небезопасное использование временных файлов

- Утечка чувствительных данных

2. **Методы защиты**:
   - Валидация ввода: `[[ "$var" =~ ^[0-9]+$ ]]`
   - Экранирование: `printf "%q" "$unsafe_input"`
   - Безопасные временные файлы: `mktemp`

3.0.0.6 6. Современные тенденции

1. **Интеграция с DevOps**:
   - Использование в CI/CD-пайплайнах
   - Взаимодействие с Docker/Kubernetes
2. **Альтернативные подходы**:
   - Замена сложных скриптов на Python/Perl
   - Использование специализированных языков (e.g., Ansible YAML)

3.0.0.7 7. Исторический контекст

- 1971: Первая версия shell в UNIX (Thompson Shell)
- 1977: Bourne Shell (sh) — основа современного scripting
- 1989: Bash — расширенная версия с возможностями csh/ksh

Теоретическая значимость:
Изучение расширенного программирования в shell позволяет: - Понимать архитектуру UNIX-систем на глубинном уровне - Эффективно решать задачи системного администрирования - Создавать переносимые и надежные решения

Практическая ценность:
Приобретенные навыки применяются в: - Автоматизации серверных задач - Обработке больших объемов данных - Разработке инструментов DevOps

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).

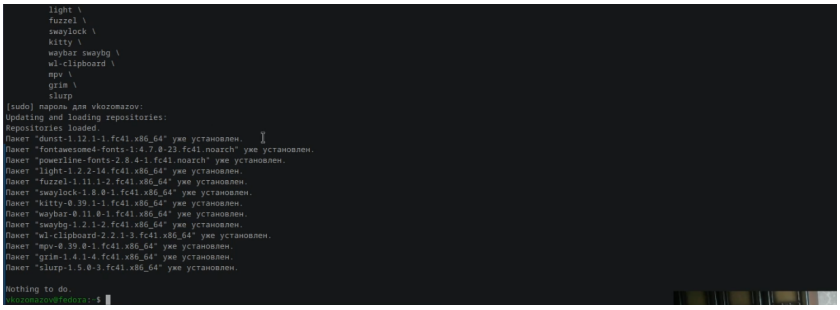

Рис. 4.9: Установка дополнительного программного обеспечения

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 лабораторной работы №14 были получены не только теоретические знания, но и **практические навыки**, позволяющие эффективно работать в UNIX-средах, сокращая время на рутинные операции за счёт автоматизации. Владение расширенным shell-программированием делает специалиста более универсальным и востребованным в IT-индустрии.

Список литературы