Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет   
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №1

на тему

**СКРИПТЫ SHELL (УГЛУБЛЕННАЯ ВЕРСИЯ)**

Выполнил: студент гр.253504 Лавренова А.С.

Проверил: ассистент кафедры информатики Гриценко Н.Ю.

Минск 2025

СОДЕРЖАНИЕ

Введение3

[1 [Ц](#_agswffz6g6ei)ель работы](#_Toc178067889) 4

[2 Теоритеческие сведения 5](#_Toc178067890)

[3 Описание и пример выполнения программы 8](#_Toc178067891)

Заключение [9](#_Toc178067899)

[Список использованных источников 10](#_Toc178067900)

[Приложение А (обязательное) 11](#_Toc178067901)

# ВВЕДЕНИЕ

В современных операционных системах скрипты shell играют важную роль в автоматизации задач, администрировании системы и обработке данных. Они позволяют создавать сложные сценарии, используя встроенные команды оболочки и внешние утилиты. Скрипты широко применяются системными администраторами, разработчиками и аналитиками для управления файлами, мониторинга процессов, автоматического развертывания приложений и выполнения рутинных операций.

Одним из ключевых преимуществ скриптов shell является их тесная интеграция с Unix-подобными системами, что позволяет эффективно использовать мощные инструменты, такие как sed, awk, wget, grep, cut, sort, а также системные утилиты для работы с процессами и файлами. Кроме того, скрипты позволяют обрабатывать данные в потоковом режиме, взаимодействовать с удалёнными серверами и упрощать сложные вычислительные задачи.

В данной лабораторной работе будут рассмотрены ключевые элементы и конструкции скриптов shell, включая переменные, параметры, ветвления, циклы, арифметические и строковые операции. Особое внимание будет уделено механизму обработки данных с использованием внешних программ и средств автоматизации, что является важным аспектом работы с Unix-системами.

1 ЦЕЛЬ РАБОТЫ

Целью данной работы является приобретение навыков написания сложных скриптов shell для решения практических задач, а также изучение принципов интеграции программ Unix с помощью сценариев оболочки. В ходе выполнения работы потребуется ознакомиться с методами генерации случайных чисел, динамическим обновлением интерфейсов в консоли и оптимизацией работы скриптов с точки зрения производительности.

В качестве практического задания предлагается разработать скрипт, реализующий консольный хранитель экрана в виде текстовых часов, изменяющих свое положение на экране. Данный проект потребует использования генераторов случайных чисел для вычисления координат отображения, механизмов периодического обновления экрана, а также работы с временными задержками и обработкой сигналов. Важным аспектом реализации является обеспечение плавного обновления положения часов через заданные промежутки времени, а также учет особенностей различных оболочек (bash, zsh, csh и др.).

Выполнение данной лабораторной работы позволит студентам не только углубить знания в области программирования на shell, но и получить ценные навыки, применимые в реальной практике системного администрирования и автоматизации.

2 ТЕОРИТИЧЕСКИЕ СВЕДЕНИЯ

Linux — это свободная операционная система (точнее, семейство систем) с открытым исходным кодом. Само название Linux относится к ядру системы, которое является ее ключевым компонентом, управляющим центральным процессором (ЦП), памятью и периферийными устройствами компьютера. [1]

Сценарии командной строки — это наборы тех же самых команд, которые можно вводить с клавиатуры, собранные в файлы и объединённые некоей общей целью. При этом результаты работы команд могут представлять либо самостоятельную ценность, либо служить входными данными для других команд. Сценарии — это мощный способ автоматизации часто выполняемых действий.

Рассмотрим структуру скрипта Shell:

1. Шебанг (Shebang**)**: #!/bin/bash - это строка в начале скрипта, указывающая на то, какой интерпретатор использовать. В данном случае используется интерпретатор Bash.
2. Комментарии: комментарии помогают описать, что делает скрипт. Они начинаются с символа # и игнорируются интерпретатором.
3. Объявление переменных: здесь объявляются переменные VAR1 и VAR2, которые содержат строки «Hello» и «World» соответственно.
4. Основная часть скрипта: это место, где выполняются основные действия. В данном случае, скрипт просто выводит содержимое переменных с помощью команды echo.
5. Завершение скрипта: команда exit 0 указывает на успешное завершение скрипта. Код 0 обычно означает успешное выполнение, а другие значения могут указывать на различные ошибки или условия завершения.

Переменные в shell играют ключевую роль, позволяя хранить и использовать данные в скриптах.

1. Объявление переменных: переменные объявляются без использования ключевых слов, просто путем присваивания значения.
2. Использование переменных: для использования переменной в тексте скрипта нужно указать её имя, предваряя его символом доллара $.
3. Специальные переменные: в shell есть множество специальных переменных, таких как $0, $1, $2, и т.д., которые хранят аргументы командной строки.
4. Чтение значений из ввода: можно использовать команду read, чтобы прочитать значение из ввода и присвоить его переменной.
5. Удаление переменных: для удаления переменной используется команда unset.
6. Интерполяция переменных: переменные могут быть интерполированы в строковых значениях.
7. Переменные окружения: все переменные, которые вы экспортируете из shell, становятся переменными окружения для любых запущенных подкоманд или программ.
8. Именованные переменные: в bash и других расширениях shell можно использовать ассоциативные массивы для хранения данных.

Переменные окружения представляют собой переменные, которые могут быть установлены в системе операционной или виртуальной среде и доступны для всех процессов, запущенных в этой среде. Они могут содержать различную информацию о системе, пользователе, конфигурации и многое другое. Эти переменные можно использовать в shell скриптах для доступа к информации о среде выполнения и для настройки поведения программы в соответствии с окружением. Для просмотра всех переменных окружения в вашей текущей среде выполнения вы можете использовать команду env в большинстве Unix-подобных систем. [2]

Операторы ветвления в shell используются для принятия решений на основе условий и выполнения различных действий в зависимости от результатов этих условий.

* if-then-else;
* if-then (без else);
* if-elif-else;
* case.

Эти операторы ветвления позволяют создавать гибкие скрипты, которые могут выполнять различные действия в зависимости от условий, и обеспечивают управление потоком выполнения программы в shell.

В контексте программирования “предикат” обычно означает выражение, которое оценивается как истинное или ложное. В языках программирования, включая shell, предикаты используются в операторах ветвления (например, в операторах if, while, case и других) для принятия решений на основе условий.

В shell скриптах часто используются циклы для повторения определенных действий определенное количество раз или до выполнения определенного условия.

* цикл while;
* цикл until;
* цикл for (для перебора элементов в списке);
* цикл for (для перебора элементов в массиве или списке);
* цикл select (для создания интерактивного меню).

Эти циклы могут использоваться для автоматизации повторяющихся задач, обработки данных, итерации по спискам и многих других вещей в shell скриптах.

Вызов сторонних программ из shell скриптов - обычная практика, которая позволяет скриптам взаимодействовать с другими программами и выполнять различные задачи. Для вызова сторонней программы используются команды, которые могут быть исполнены внутри скрипта. [3]

Один из основных способов интеграции Unix-программ — это прямое использование команд в скриптах. В Unix-системах есть множество стандартных утилит и команд, которые выполняют различные операции, такие как манипуляция файлами, работа с текстом, управление процессами и т. д. Пример таких команд: ls, grep, awk, sed, cp, mv, find, sort, и многие другие.

В Unix существует мощная концепция пайпов, которая позволяет передавать вывод одной программы на вход другой программы. Это позволяет интегрировать несколько Unix-

Многие Unix-программы могут работать с данными, поступающими через стандартный ввод (stdin), и передавать результаты через стандартный вывод (stdout). Эти особенности можно использовать для интеграции программ в рамках скриптов.

В Unix-системах широко используются переменные окружения для хранения конфигурации и других значений, доступных для различных программ. Эти переменные можно устанавливать и использовать внутри скриптов.

В Unix можно запускать программы в фоновом режиме, что позволяет интегрировать их выполнение в скрипты, не блокируя выполнение других команд. Для этого используется

Можно автоматизировать выполнение различных утилит и программ через скрипты, позволяя интегрировать их с другими процессами или выполнять заранее определённые задачи на основе условий.

Важно, чтобы интеграция программ в скриптах предусматривала обработку ошибок. В Unix каждой программы, запущенной в скрипте, есть статус завершения (exit status), который можно использовать для принятия решений о дальнейших действиях.

В Bash и других оболочках можно создавать функции, которые упрощают повторное использование кода и делают интеграцию с другими программами более удобной. Это позволяет собирать комплексные операции в отдельные блоки, которые могут быть использованы в разных частях скрипта.

Хотя Linux в основном используется технологически продвинутыми пользователями, она является такой же операционной системой, как более популярные Windows и MacOS, и обладает похожим набором функций. Например, Linux имеет графический интерфейс пользователя и собственные версии часто используемых программ, таких как фоторедакторы, электронные таблицы и почтовые клиенты. Linux можно установить на любое электронное устройство, от компьютера до смартфона. Однако у этой операционной системы есть и отличия. Именно поэтому Linux может быть использована в иных целях, чем Windows и MacOS.

**3 ОПИСАНИЕ И ПРИМЕР ВЫПОЛНЕНИЯ ПРОГРАММЫ**

Разработанный скрипт представляет собой консольный хранитель экрана, отображающий цифровые часы, которые перемещаются по экрану через фиксированные промежутки времени (рисунок 3.1).

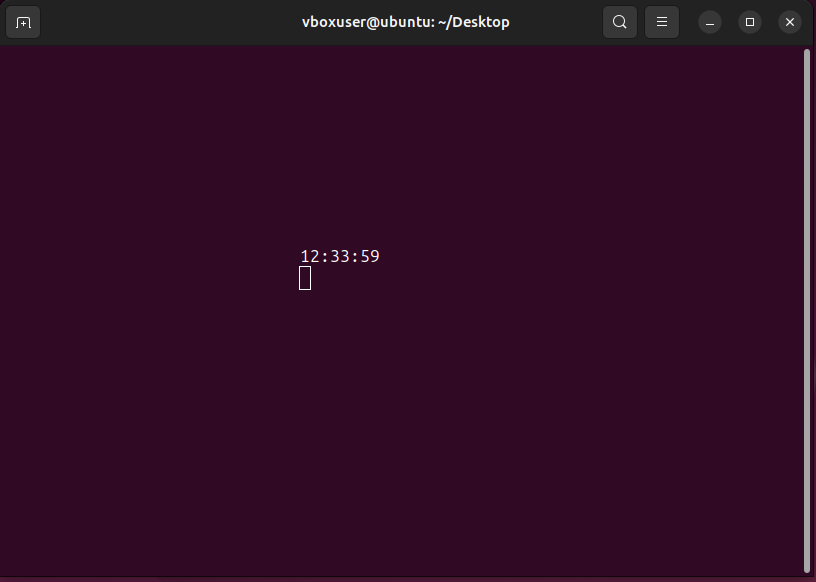


Рисунок 3.1 – Терминал с результатом выполнения

Основные функции программы включают:

1. Программа определяет размеры текущего окна терминала с помощью команд tput lines и tput cols.
2. Генерация случайных координат для отображения часов выполняется функцией random\_position(), использующей встроенную переменную $RANDOM. Алгоритм учитывает размеры терминала и обеспечивает корректное размещение часов.
3. Каждые 10 секунд часы перемещаются в новую случайную позицию, используя функцию random\_position().
4. Проверка выполняется с помощью переменной SECONDS\_ELAPSED, которая отслеживает прошедшее время.
5. В цикле while true программа очищает экран (clear).
6. Получает текущее системное время с помощью команды date +"%H:%M:%S".
7. Выводит часы в ранее вычисленные координаты с использованием tput cup. Использование tput cup позволяет разместить текст в нужных координатах, избегая перерисовки всего экрана.
8. Каждую секунду программа обновляет время на экране (sleep 1).

Таким образом, программа реализует текстовый хранитель экрана в виде перемещающихся цифровых часов, используя элементы управления терминалом и встроенные механизмы генерации случайных чисел.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были изучены основные элементы и конструкции скриптов shell, а также их практическое применение для создания консольного хранителя экрана в виде перемещающихся текстовых часов. Разработанный скрипт продемонстрировал возможности автоматизации и динамического обновления информации в терминале, используя встроенные команды оболочки и системные утилиты.

Одной из ключевых задач работы стала генерация случайных координат для отображения часов. Для этого использовался встроенный генератор случайных чисел в сочетании с параметрами размеров терминала, получаемыми через tput. Это позволило гибко адаптировать положение часов к текущему размеру консоли. Также была реализована периодическая смена координат, что соответствует концепции экранного хранителя.

Лабораторная работа продемонстрировала, что скрипты shell обладают высокой гибкостью и возможностью интеграции с системными инструментами, что делает их мощным средством автоматизации в Unix-подобных системах. Использование встроенных команд, управления позиционированием текста в терминале с помощью tput, а также работы с системными утилитами подтверждает актуальность shell-скриптов для решения различных задач администрирования и автоматизации.

Таким образом, в результате выполнения лабораторной работы были получены практические навыки работы с shell-скриптами, изучены методы обработки данных и взаимодействия с терминалом, что является важным аспектом для дальнейшего освоения системного администрирования и программирования в среде Unix.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Linux и UNIX: программирование в shell. Руководство разработчика. Дэвид Тейнсли [Электронный ресурс]. – Режим доступа: https://coollib. com/b/275969/read?ysclid=m6ujdy2lag999562018.

[2] Язык командного энтерпритатора Shell. Бизюк Андрей [Электронный ресурс]. – Режим доступа: https://it.vstu.by/courses/information\_systems/syst em\_prog ramming/practice/shell/.

[3] ПрограммированиенаBashснуля Илья Шпигорь – [Электронный ресурс]. – Режим доступа: https://ugolok.vercel.app/books/bash.pdf.

ПРИЛОЖЕНИЕ А

(обязательное)

Исходный код

LINES=$(tput lines)

COLUMNS=$(tput cols)

random\_position() {

echo $(( RANDOM % (LINES - 6) + 1 )) $(( RANDOM % (COLUMNS - 20) + 1 ))

}

read ROW COL <<< $(random\_position)

SECONDS\_ELAPSED=0

while true; do

clear

if (( SECONDS\_ELAPSED % 10 == 0 )); then

read ROW COL <<< $(random\_position)

fi

TIME=$(date +"%H:%M:%S")

tput cup $ROW $COL

tput cup $((ROW+1)) $COL

echo "$TIME "

tput cup $((ROW+2)) $COL

sleep 1

((SECONDS\_ELAPSED++))

done