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

**ОТЧЕТ**

к лабораторной работе № 1

на тему

**СКРИПТЫ SHELL**

Выполнил             П. Н. Носкович

Проверил                          Н. Ю. Гриценко

Минск 2025

СОДЕРЖАНИЕ

[Введение 3](#_Toc158758843)

[1 Краткие теоретические сведения 4](#_Toc158758844)

[2 Результаты выполнения лабораторной работы 7](#_Toc158758845)

[Заключение 8](#_Toc158758846)

[Список использованных источников 9](#_Toc158758847)

[Приложение А](#_Toc158758848) [(обязательное)](#_Toc158758849) [Листинг кода 10](#_Toc158758850)

## ВВЕДЕНИЕ

Цель данной лабораторной работы заключается в освоении различных элементов и конструкций оболочки *shell*, таких как переменные, параметры, операторы ветвлений, циклы, вычисления и вызовы внешних приложений, для решения сложной практической задачи. Кроме того, необходимо изучить способы интеграции программ *Unix* с помощью *shell*-скриптов. Требуется разработать и реализовать скрипт для создания консольного хранителя экрана (*screensaver*) – большие текстовые часы, меняющие своё расположение на экране.

В рамках выполнения данной работы предполагается детальное изучение основных возможностей *bash*, включая работу с позиционными параметрами, специальными переменными окружения и механизмами подстановки команд. Важно освоить способы обработки пользовательского ввода, управления процессами и организации фонового выполнения скриптов.

Также необходимо изучить принципы взаимодействия *bash*-скриптов с системными утилитами *Linux*, такими как *date* для получения текущего времени, tput для управления курсором в терминале и *clear* для обновления экрана.

## 1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Командный интерпретатор или оболочка (*Shell*) – программа, обеспечивающая взаимодействие с пользователем посредством интерфейса командной строки. В соответствии с командами пользователя или с интерпретируемым «сценарием» оболочка обеспечивает исполнение других программ (процессов) в нужном порядке и с нужным взаимодействием между собой. Таким образом, оболочка занимает место посредника между пользователем и системой (и другими программами).

*Bash* (*Bourne again shell*) – это модифицированная версия программной оболочки *Bourne-shell*. Она является командным процессором, работающим интерактивно в текстовом окне. *Bash* нужен для приема команд пользователя и их отправки операционной системе для последующей обработки. [1]

В качестве языка программирования *shell* обеспечивает решение специализированной задачи: выполнение других программ («задач») и управление ими. Как и любой язык программирования, он содержит в себе операторы, ключевые слова, типы данных, переменные, константы и. т. д.

Управляющие операторы – «разделители» командной строки (списка) на конвейеры. Безусловное следование – команды (конвейеры) выполняются последовательно, от первого до последнего и разделяются «;». Условное следование – следующий конвейер (команда) выполняется, если код завершения предыдущего «успешный» (т.е. =0) и разделяются «&&» или код завершения предыдущего «неуспешный» (т.е. != 0) и разделяются «||». В любом случае итоговый код завершения командной строки (списка) – код завершения последнего выполненного конвейера (команды).

Каждому процессу позволено иметь до девяти открытых дескрипторов файлов. Оболочка *bash* резервирует первые три дескриптора с идентификаторами 0 (*STDIN*), 1 (*STDOUT*) и 2 (*STDERR*). [2]

*STDIN* – это стандартный поток ввода оболочки. Для терминала стандартный ввод — это клавиатура. Когда в сценариях используют оператор перенаправления ввода «<», *Linux* заменяет дескриптор файла стандартного ввода на тот, который указан в команде. Система читает файл и обрабатывает данные так, будто они введены с клавиатуры.

*STDOUT* – стандартный поток вывода оболочки. По умолчанию это – экран. Большинство *bash*-команд выводят данные в *STDOUT*, что приводит к их появлению в консоли. Данные можно перенаправить в файл, присоединяя их к его содержимому при помощи оператора «>>», или перезаписать файл при помощи оператора «>».

*STDERR* представляет собой стандартный поток ошибок оболочки. По умолчанию этот дескриптор указывает на то же самое, на что указывает *STDOUT*, именно поэтому при возникновении ошибки мы видим сообщение на экране. Если нужно перенаправить данный поток в файл, то после команды нужно указать номер идентификатора и имя файла, например «2>*file.txt*».

С помощью команды *tput*, использующей базу данных *terminfo, shell* получает доступ к информации и характеристикам, зависящим от типа конкретного терминала. Посредством этой команды можно также инициализировать или переустановить терминал, получить длинное имя запрашиваемого типа терминала. Если атрибут с заданным именем является цепочкой символов, то команда *tput* возвращает цепочку символов; если атрибут является целым числом, то возвращается целое число. Если же атрибут логического типа, то *tput* просто устанавливает код завершения [0 (истина), если терминал имеет данную характеристику; 1 (ложь), если он ее не имеет] и ничего не выводит.

Основной тип – строка. В некоторых случаях (некоторыми командами) строка интерпретируется как число. Некоторые интерпретаторы предусматривают числовые типы (например, *csh*). Как правило, ограничиваются целыми числами с фиксированной разрядностью.

Массивы поддерживаются некоторыми интерпретаторами (в том числе *bash*), но не *sh*. В качестве индекса могут выступать константы и переменные, а также специальное значение «@» – обращение ко всему списку значений.

Переменная, значение которой не задано при объявлении, получает пустое значение. При попытке использования необъявленной заранее переменной также будет подставлено пустое значение, ошибка произойдет только если пустое значение недопустимо в конкретном контексте.

Созданные переменные включаются в окружение интерпретатора. Каждый процесс получает при создании блок окружения и в дальнейшем может пополнять его новыми переменными, которые становятся равноправны с унаследованными (кроме передачи порожденным процессам).

Необходимо всегда помнить о различиях между именем переменной и ее значением. Если *variable1* – это имя переменной, то $*variable1* – это ссылка на ее значение. Имена переменных, без префикса $, могут использоваться только при объявлении переменный, при присваивании переменной некоторого значения, при удалении (сбросе) и при экспорте. [3]

Объявленные переменные, в отличие от унаследованного окружения, не передаются порожденным процессам. Чтобы обеспечить такую передачу, используется команда *export*. В любом случае, порожденным процессам переменные передаются исключительно по значению. Изменения в окружении процесса-потомка не влияют на процесс-родитель.

Виды циклов: цикл *while* с прямым условием (т. е. выполняется, пока код возврата команды-условия равняется 0), цикл *until* с инверсным условием (т. е. выполняется, пока код возврата команды-условия равняется 1) и цикл *for*. Для принудительного безусловного выхода из цикла используется ключевое слово *break*. Для принудительного перехода на следующую итерацию используется ключевое слово *continue*. Для завершения текущего сценария или сеанса *shell* используется ключевое слово *exit*.

Оператор *if/then* проверяет, является ли код завершения списка команд 0, и если это так, то выполняет команды, следующие за словом *then*.

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы была разработана программа для создания консольного экранного сейвера, который отображает большие текстовые часы, перемещающиеся в случайные места экрана. Программа обновляет время каждую секунду и меняет положение часов через фиксированные промежутки времени, используя случайные координаты.

Для генерации случайных позиций использовался алгоритм, основанный на текущем времени. Это позволяет делать положение часов на экране динамичным и непредсказуемым. Пример работы программы можно увидеть на рисунке в консоли, где отображаются часы, меняющие своё местоположение в случайном порядке.

При запуске программа начинает отображать текущее время в крупном формате с использованием утилиты *figlet*. Время обновляется каждую секунду, а через определённое количество обновлений (в данном случае каждые две секунды) часы перемещаются в новую случайную позицию на экране.

Для генерации случайных координат используется алгоритм, основанный на текущем времени. Это делает расположение часов динамичным и непредсказуемым, поскольку каждое новое вычисление координат зависит от изменяющегося во времени значения.

При запуске программы сначала определяется текущее время, после чего оно преобразуется в текстовый формат, например: *twelve HOURS thirty MINUTES forty-five SECONDS*. Это достигается с помощью функции, которая разбивает время на часы, минуты и секунды и переводит каждое числовое значение в словесную форму. Затем вычисляются случайные координаты, на которые будет перемещён курсор консоли перед выводом текста. Для этого используются специальные функции, реализующие генерацию псевдослучайных чисел на основе текущего времени.

Основной цикл программы выполняется бесконечно: каждую секунду экран очищается, часы обновляются, а через каждые два обновления они перемещаются в новое случайное место. Таким образом создаётся эффект плавающих часов, напоминающий классические экранные заставки.

## ЗАКЛЮЧЕНИЕ

В результате выполнения данной лабораторной работы была успешно разработана и реализована программа, выполняющая функцию экранного сейвера для консоли. Этот сейвер отображает большие текстовые часы, которые каждый раз перемещаются в случайное место на экране через фиксированные интервалы времени (каждые *t* секунд).

Реализация программы включала использование оболочки *shell*, что позволило эффективно работать с переменными, циклами и случайными числами. Основной задачей было создать динамичное и интересное отображение времени, что было достигнуто за счёт перемещения текста с часами в разные части экрана с помощью случайной генерации координат.

Как результат, был реализован не только функционал отображения времени, но и добавлен элемент случайности, который делает процесс более живым и визуально привлекательным для пользователя.

Одним из важных аспектов данной работы является её универсальность и возможность дальнейшего расширения. Разработанный алгоритм может быть адаптирован для различных целей, например, для создания более сложных экранных заставок, отображения дополнительной информации или интеграции с другими системами мониторинга. Кроме того, концепция случайного размещения текста может быть использована в других проектах, связанных с динамическим выводом данных в консоли.

В ходе выполнения лабораторной работы был не только реализован функционал по отображению текущего времени, но и добавлен элемент случайности, который делает процесс более живым и визуально привлекательным. Полученный результат демонстрирует возможности *Bash*-скриптов в создании интерактивных консольных приложений и показывает, как с помощью простых инструментов можно добиться интересного визуального эффекта.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Основы работы с командной оболочкой *bash* в *Linux* [Электронный ресурс]. – Режим доступа: https://eternalhost.net/blog/sistemnoeadministrirovanie/linux-bash-chto-eto – Дата доступа: 13.02.2025

[2] Программирование в *shell* [Электронный ресурс]. – Режим доступа: <https://studfile.net/preview/919423/page:26/> – Дата доступа: 13.02.2024

[3] Переменные и параметры. Введение. [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/bash\_scripting\_guide/c1613.html – Дата доступа: 13.02.2025

[4] Проверка условий [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/bash\_scripting\_guide/c2171.html – Дата доступа: 13.02.2025

# **ПРИЛОЖЕНИЕ А**

## (обязательное)

## Листинг кода

#!/bin/bash

convert\_current\_time\_to\_words(){

local current\_time=$(date +%T)

echo "$(convert\_to\_words "${current\_time:0:2}") HOURS $(convert\_to\_words "${current\_time:3:2}") MINUTES $(convert\_to\_words "${current\_time:6:2}") SECONDS"

}

convert\_to\_words() {

local num=$1

case $num in

00) echo "o" ;;

01) echo "one" ;;

02) echo "two" ;;

03) echo "three" ;;

04) echo "four" ;;

05) echo "five" ;;

06) echo "six" ;;

07) echo "seven" ;;

08) echo "eight" ;;

09) echo "nine" ;;

10) echo "ten" ;;

11) echo "eleven" ;;

12) echo "twelve" ;;

13) echo "thirteen" ;;

14) echo "fourteen" ;;

15) echo "fifteen" ;;

16) echo "sixteen" ;;

17) echo "seventeen" ;;

18) echo "eighteen" ;;

19) echo "nineteen" ;;

2[0-9]) echo "twenty $(convert\_to\_words "0${num:1}")" ;;

3[0-9]) echo "thirty $(convert\_to\_words "0${num:1}")" ;;

4[0-9]) echo "forty $(convert\_to\_words "0${num:1}")" ;;

5[0-9]) echo "fifty $(convert\_to\_words "0${num:1}")" ;;

\*) echo "$(convert\_to\_words "${num:1}")" ;;

esac

}

print\_at\_coordinates() {

local x=$1

local y=$2

local message=$(convert\_current\_time\_to\_words)

echo -e "\e[${y};${x}H${message}"

}

# Y must be [0;48] in full screen

random\_number\_generator\_y(){

local current\_time=$(date "+%H%M%S%3N")

local a=226954

local c=1

local m=49

echo $(( ($a\*$current\_time + c) % m ))

}

# X must be [39; 364] in full screen

random\_number\_generator\_x(){

local current\_time=$(date "+%H%M%S%3N")

local a=226954

local c=1

local m=326

echo $(( (($a\*$current\_time + c) % m) + 39 ))

}

print\_time(){

local counter=0

local x=$(random\_number\_generator\_x)

local y=$(random\_number\_generator\_y)

while true

do

clear

print\_current\_time\_with\_figlet $x $y

((counter++))

if [ $counter -eq 2 ]

then

x=$(random\_number\_generator\_x)

y=$(random\_number\_generator\_y)

counter=0

fi

sleep 1s

done

}

print\_current\_time\_with\_figlet(){

local x=$1

local y=$2

local current\_time=$(date +%T)

tput cup $y

figlet -w $x -c $current\_time

}

clear

print\_time