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             В.П. Бычко

Проверил                          Н. Ю. Гриценко

Минск 2024

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

Целью выполнения данной лабораторной работы является изучение элементов и конструкций скриптов shell: переменных, параметров, ветвлений, циклов, вычислений и вызовов внешних программ для решения достаточно сложной задачи, имеющей практическое значение, а также принципов интеграции Unix-программ скриптами shell. Разработать и реализовать хранитель экрана (screensaver) для консоли – большие текстовые часы, меняющие своё расположение на экране

## 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.

Существует специальная команда – «[» (левая квадратная скобка). Она является синонимом команды test, и является встроенной. Эта команда воспринимает свои аргументы как выражение сравнения или как файловую проверку и возвращает код завершения в соответствии с результатами проверки (0 – истина, 1 – ложь).

Bash предоставляет в распоряжение программиста конструкцию [[ ... ]] расширенный вариант команды test, которая выполняет сравнение способом более знакомым программистам, пишущим на других языках программирования. Обратите внимание: « [[» – это зарезервированное слово, а не команда. [4]

Таким образом, bash – это мощный и гибкий интерпретатор командной строки, широко используемый в UNIX-подобных операционных системах. Он является стандартной оболочкой в большинстве Linux-дистрибутивов и часто используется в качестве основного инструмента для взаимодействия с операционной системой и выполнения различных задач. А также имеет следующие особенности: гибкость, скриптовый язык, переменные и условные конструкции, команды и утилиты.

# 2 РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ЛАБОРАТОРНОЙ РАБОТЫ

В ходе выполнения лабораторной работы был реализован хранитель экрана (screensaver) для консоли – большие текстовые часы, которые перемещаются в случайное место на экране раз в 10 секунд. На рисунке 3.1 представлен пример работы программы.



Рисунок 3.1 – Пример работы программы

## ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы был разработан и реализован хранитель экрана (screensaver) для консоли – большие текстовые часы, которые перемещаются в случайное место на экране раз в 10 секунд.

## 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: 13.02.2024

[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: 13.02.2024

[4] Проверка условий [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/bash\_scripting\_guide/c2171.html – Дата доступа: 13.02.2024

# **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