Санкт-Петербургский национальный исследовательский университет
информационных технологий, механики и оптики

Факультет инфокоммуникационных технологий

Направление подготовки 11.03.02

Лабораторная работа №2

Создание и использование размерных типов данных

Выполнил:

Дощенников Никита Андреевич

Группа: К3221

Проверил:

Береснев Артем Дмитриевич

Санкт-Петербург

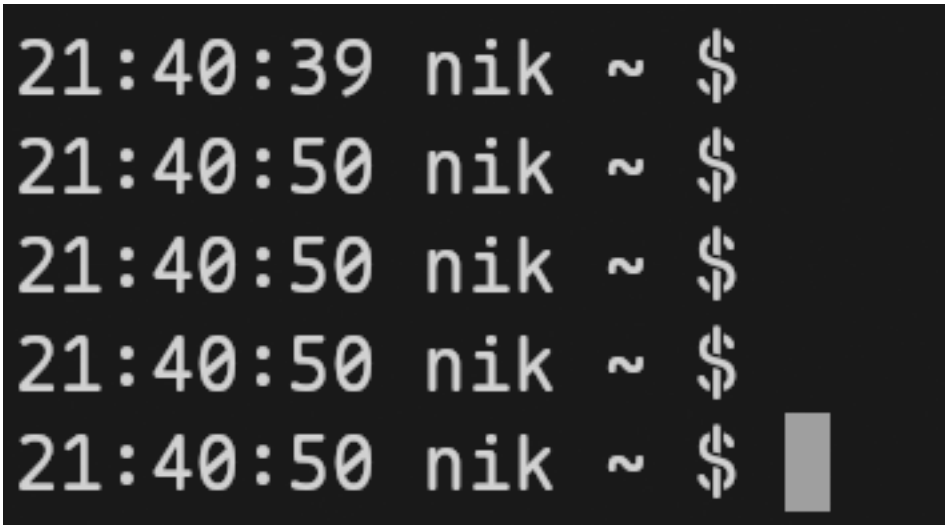
2025

Цель работы:
Получить практические навыки программирования на bash, освоить практические приемы работы с командной оболочкой bash.

Часть 1. Работа в среде bash.
Для изменения приглашения командной строки я отредактировал файл `~/.bashrc`, добавив туда строку:

```
PS1='\t \u \w \$ '
```

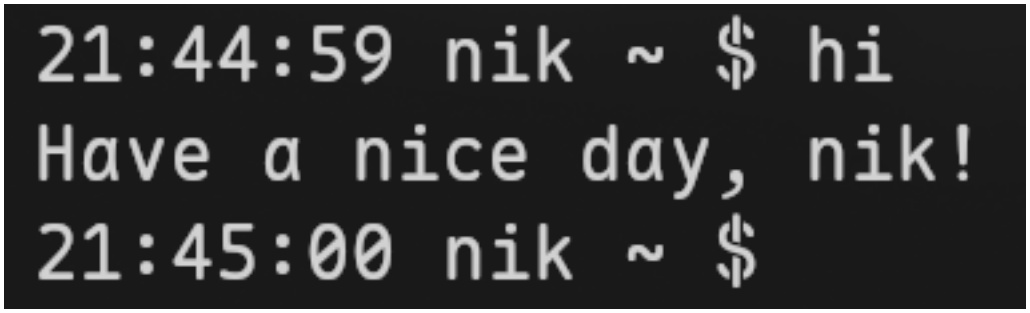
Теперь приглашение отображает текущее время, имя пользователя, текущий каталог и символ `$`.



Создал алиас для вывода приветствия:

```
alias hi='echo "Have a nice day, $(whoami)!"'
```

После этого при вводе команды `hi` в терминале появляется строка с приветствием и именем текущего пользователя.



Чтобы изменения сохранились при перезапуске, эти строки были добавлены в файл `~/.bashrc`. Применил изменения без выхода из системы командой:

```
source ~/.bashrc
```

Часть 2. Простые задачи на bash.

1. Принимает две строки как параметры и сравнивает их.

```
#!/bin/bash
[ "$#" -ne 2 ] && echo "usage: $0 str1 str2" && exit 1
[ "$1" = "$2" ] && echo "equal" || echo "not equal"
```

```
21:58:16 nik ~/linux_intro/labs/lab2/scripts $ script221.sh foo bar
not equal
21:58:21 nik ~/linux_intro/labs/lab2/scripts $ script221.sh bar bar
equal
```

2. Принимает три целых числа и находит максимальное из них.

```
#!/bin/bash
[ "$#" -ne 3 ] && echo "usage: $0 a b c" && exit 1
a=$1
b=$2
c=$3
max=$a
(( b > max )) && max=$b
(( c > max )) && max=$c
echo "$max"
```

```
22:00:55 nik ~/linux_intro/labs/lab2/scripts $ script222.sh 1 2 3
3
```

3. Считывает строки до тех пор, пока не встретится строка `q`. После завершения цикла все введённые строки объединяются и выводятся в одну строку.

```
#!/bin/bash
arr=()
while IFS= read -r line; do
```

```
  [ "$line" = "q" ] && break
  arr+=("$line")
done
printf "%s" "${arr[*]}"
```

```
22:08:40 nik ~/linux_intro/labs/lab2/scripts $ script223.sh
f
o
o
b
a
r
q
f o o b a r 22:08:48 nik ~/linux_intro/labs/lab2/scripts $
```

4. Считывает последовательность целых чисел. Чтение продолжается, пока не встретится чётное число. После этого выводится количество введённых чисел до него.

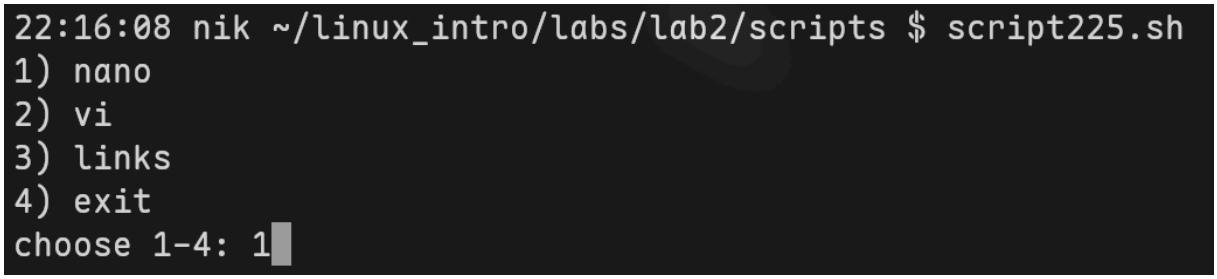
```
#!/bin/bash
count=0
while IFS= read -r n; do
  r=$(( n % 2 ))
  [ "$r" -eq 0 ] && break
  count=$(( count + 1 ))
done
echo "$count"
```

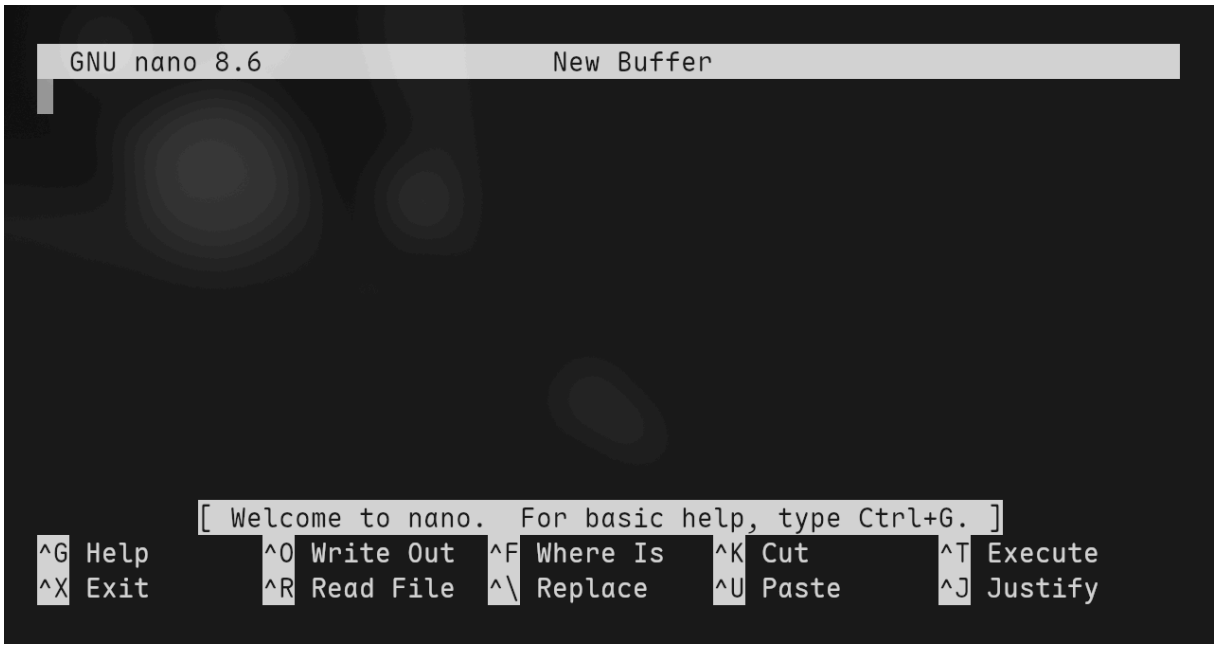
```
22:10:13 nik ~/linux_intro/labs/lab2/scripts $ script224.sh
1
3
5
7
2
4
22:10:23 nik ~/linux_intro/labs/lab2/scripts $
```

5. Создано текстовое меню с четырьмя пунктами: запуск редактора `nano`, редактора `vi`, браузера `links`, или выход. Перед выполнением команда `command -v` проверяет наличие программ.

Если программа не установлена — выводится сообщение и скрипт завершается.

```
#!/bin/bash
for cmd in nano vi links; do
  command -v "$cmd" >/dev/null 2>&1 || { echo "$cmd is not
installed"; exit 1; }
done
echo "1) nano"
echo "2) vi"
echo "3) links"
echo "4) exit"
read -r -p "choose 1-4: " choice
case "$choice" in
  1) exec nano ;;
  2) exec vi ;;
  3) exec links ;;
  4) exit 0 ;;
  *) echo "invalid choice" ; exit 1 ;;
esac
```





6. Принимает три параметра: имя файла, режим (`h` или `t`), и число строк. В режиме `h` печатаются первые N строк файла (`head -n`). В режиме `t` — последние N строк (`tail -n`). Ошибки при отсутствии файла подавляются перенаправлением `2>/dev/null`.

```
#!/bin/bash
[ "$#" -ne 3 ] && echo "usage: $0 file h|t N" && exit 1
file=$1
mode=$2
num=$3
case "$mode" in
  h) head -n "$num" -- "$file" 2>/dev/null ;;
  t) tail -n "$num" -- "$file" 2>/dev/null ;;
  *) echo "mode must be h or t" ; exit 1 ;;
esac
```

```
22:10:13 nik ~/linux_intro/labs/lab2/scripts $ script224.sh
1
3
5
7
2
4
22:10:23 nik ~/linux_intro/labs/lab2/scripts $
```

Часть 3. Разработка скрипта расчета производительности.

1. В этой части требовалось написать скрипт `script23`, который измеряет производительность CPU в FLOPS (Floating Point Operations Per Second). Для этого необходимо было реализовать большое количество операций с плавающей запятой и зафиксировать время их выполнения.

2. Для вычислений использовалась утилита `bc`, так как она позволяет работать с произвольной точностью. В скрипте выставлена точность `scale=10`.

3. В скрипте организована функция `bench`, в которой выполняется N умножений двух одинаковых чисел. Параметр N можно передать в скрипт, по умолчанию используется 1 000 000.

4. С помощью команды `date +%s.%N` время сохраняется до и после выполнения функции. Далее определяется разница во времени и вычисляется показатель FLOPS по формуле:

$$FLOPS = \frac{N}{\Delta t}$$

5. В завершении программа выводит три значения: количество итераций, время выполнения в секундах, и рассчитанный FLOPS.

```
#!/bin/bash
n=${1:-1000000}
command -v bc >/dev/null 2>&1 || { echo "bc is required"; exit
1; }

bench() {
  printf
'scale=10\na=1.2345678901\nb=1.2345678901\nfor(i=0;i<%s;i++)
x=a*b\n' "$n" | bc >/dev/null
}

start=$(date +%s.%N)
bench
end=$(date +%s.%N)

dur=$(echo "$end - $start" | bc -l)
flops=$(echo "$n / $dur" | bc -l)
```

```
printf "iterations=%s\ntime_s=%s\nFLOPS=%s\n" "$n" "$dur"
"$flops"
```

```
22:30:13 nik ~/linux_intro/labs/lab2/scripts $ script23.sh
iterations=1000000
time_s=.400653495
FLOPS=2495922.318111813800600940 72
22:30:16 nik ~/linux_intro/labs/lab2/scripts $
```

Вопросы.

1. Как определить сколько параметров пользователь передал в скрипт при запуске?
В bash для этого используется специальная переменная `$#`. Она хранит количество аргументов, переданных скрипту. Пример:

```
echo "amount of params: $#"
```

2. Что выведет простой скрипт с постдекрементом?
Код:

```
count=10
echo $(( count-- ))
echo $(( count-- ))
```

```
22:35:26 nik ~/linux_intro/labs/lab2/scripts $ q.sh
10
9
```

Первая строка напечатает `10`, потому что оператор постдекремента возвращает старое значение, а потом уменьшает переменную. После этой операции `count=9`. Вторая строка напечатает `9` — аналогично.

3. Недостатки подхода к определению FLOPS в нашем скрипте

- Измеряется не только операция умножения, но и работа интерпретатора bash и программы `bc`.
- Не используется SIMD (векторные инструкции) и многопоточность, поэтому результат сильно занижен.

- Выполняется только умножение одинаковых чисел, что не отражает реальную работу CPU.
- На результат влияют другие процессы в системе, частота процессора, троттлинг, кэш.
- Такой метод больше демонстрационный и не сопоставим с эталонными тестами на C/Fortran.

Вывод.

В ходе проделанной работы я получил практические навыки программирования на bash, освоил практические приемы работы с командной оболочкой bash.