Лабораторная работа №7

Starovoytov Alexandr

Цель работы

Получение навыков написания сценариев на «скриптовых» языках.

Скриптовый язык, на котором будет выполняться лабораторная работа, студентом выбирается самостоятельно. Примеры возможных скриптовых языков: JavaScipt (Node.js), Python, Ruby, Lua, Perl, Racket и т.д.

Задания

При демонстрации результатов работы преподавателю все скрипты должны запускаться командой, содержащей только имя скрипта (т.е. без указания в командной строке пути к скрипту и интерпретатора), то есть так:

```
myscript arg1 arg2
```

а не так:

```
bash ./myscript.sh arg1 arg2
```

1. На Bash напишите скрипт, который будет запускать долго выполняющуюся программу (напишите скрипт, имитирующий такую программу, скажем, просто ожидающий несколько минут и завершающийся) строго каждые *t* минут, но так, чтобы одновременно выполнялось не более 1 экземпляра этой программы. Путь к программе и периодичность запуска передавайте в виде аргументов командной строки. Вывод и ошибки запускаемой программы направляйте в файлы, имена этих файлов формируйте автоматически.

   ```
   #!/usr/bin/env bash

   # очень долго выполняющаяся программа
   ```

```
echo "executed"
>&2 echo "error"
sleep 120
```

```
#!/usr/bin/env bash

show_help() {
    cat << EOF
Usage: ./exec_periodically command interval
Runs command with interval period, if command still running - kills
        it
Stdout redirected to ./process_stdout
Stderr redirected to ./process_stderr
EOF
}

[ "$#" -ne 2 ] && show_help && exit 0

STDOUT_FILE="./process_stdout"
STDERR_FILE="./process_stderr"

program="$1"
interval="$2"

$program > "$STDOUT_FILE" 2> "$STDERR_FILE" &
sleep $(( "$interval" * 60 ))

while true; do
    $program >> "$STDOUT_FILE" 2>> "$STDERR_FILE" &
    sleep $(( "$interval" * 60 ))
    kill -9 "$!"
done
```

2. На Bash напишите скрипт, который принимает путь к проекту на языке C и выводит общее число непустых строк во всех файлах .c и =.h= указанного проекта. Предусмотрите рекурсивный обход вложенных папок.

```
#!/usr/bin/env bash

show_help() {
    cat << EOF
Usage: ./clines [-h] [path...]
```

```
Count nonempty lines in C source code files at paths

-h, --help    Display this message
EOF
}

count_nonempty_lines() {
    grep -cv '^$' "$1"
}

[ "$1" = "--help" ] || [ "$1" = "-h" ] && show_help && exit 0

paths=( "$@" )
[ "${paths[*]}" ] || paths="."

programs=$(find "${paths[@]}" -name '*.c' -o -name '*.h')
[ ! "$programs" ] && echo "C source files not found in $paths" 1>&2
        && exit 1

ans=0
while read -r file; do
    (( ans += $(count_nonempty_lines "$file") ))
done <<< "$programs"
echo "$ans"
```

3. На выбранном скриптовом языке напишите программу, которая выводит в консоль указанное число строк заданной длины, состоящих из латинских букв, цифр и печатных знаков, присутствующих на клавиатуре. Длину строки и число строк передавайте как аргументы командой строки. Для каких целей можно использовать такую программу? Оформите логику приложения в виде отдельной функции и поместите её в отдельный модуль.

    - Простой генератор паролей на Raku

      ```
      #!/usr/bin/env raku

      sub gen-strings (Int $count, Int $len --> Array:D) {
          my Str @res;
          for 1..$count {
              my Str $pass ~= ('!' .. '~').pick() for 1..$len;
              @res.push($pass);
          }
      ```

```
        return @res;
    }

    sub MAIN(
        Int $length,
        Int $count
    ) {
        for gen-strings($count, $length) -> $str {
            say $str;
        }
    }
```

4. **Задание повышенной сложности.** На выбранном скриптовом языке напишите функцию, которая принимает произвольную чистую функцию с переменным числом аргументов и возвращает мемоизованную версию этой функции. Для запоминания результатов вычислений выберете подходящую структуру данных из числа встроенных классов выбранного языка.

```
#!/usr/bin/env raku

sub memoize-function($function) {
    my %known-results;
    return -> |args {
        if %known-results{args}:exists {
            %known-results{args};
        } else {
            my $res = $function(|args);
            %known-results{args} = $res;
            $res;
        }
    }
}

sub test($a, $b) {
    say "computed";
    return $a + $b;
}

my $test-memo = memoize-function(&test);
say $test-memo(1, 2);
```

```
say $test-memo(1, 2);
say $test-memo(1, 3);
```

```
computed
3
3
computed
4
```