

## Лабораторная работа 2.18

Тема: Работа с переменными окружения в Python3.

Цель работы: приобретение навыков по работе с переменными окружения с помощью языка программирования Python версии 3.x.

### Порядок выполнения работы


1. Создаём аккаунт в GitHub. Затем создаём новый общедоступный репозиторий, в котором будет использована лицензия MIT и язык программирования Python.

#### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (\*).

Owner \*

 RomanGorchakov ▾

Repository name \*

/ Test

✓ Test is available.

Great repository names are short and memorable. Need inspiration? How about [automatic-octo-chainsaw](#) ?

Description (optional)

☒  Public

Anyone on the internet can see this repository. You choose who can commit.

☐  Private

You choose who can see and commit to this repository.

Initialize this repository with:

☐ Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: Python ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

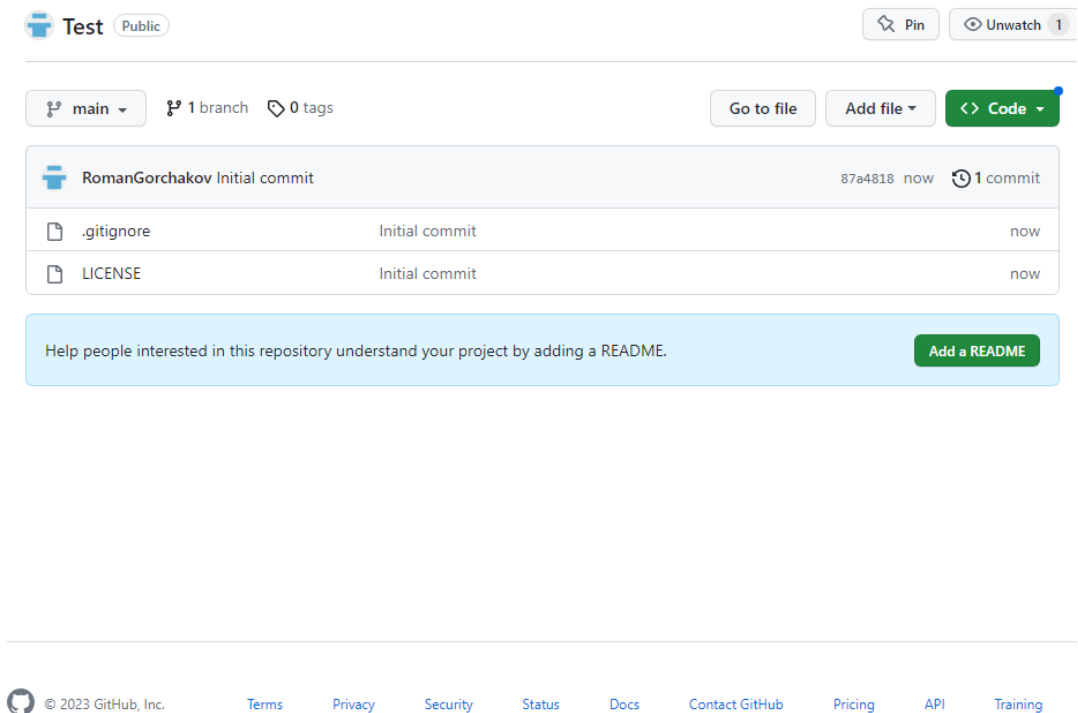
Choose a license

License: MIT License ▾


















A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

Create repository



2. Теперь необходимо дополнить файл `.gitignore` с необходимыми правилами для языка программирования Python. Для этого переходим по ссылке «<https://github.com/github/gitignore>» и скачиваем оттуда файл «Python.gitignore».

	Phalcon.gitignore	Remove trailing asterisks in Phalcon rules	10 years ago
	PlayFramework.gitignore	Added ./project/project to PlayFramework.gitignore	7 years ago
	Plone.gitignore	Covered by global vim template	10 years ago
	Prestashop.gitignore	Update for Prestashop 1.7 (#3261)	3 years ago
	Processing.gitignore	Ignore transpiled .java and .class files (#3016)	4 years ago
	PureScript.gitignore	Update PureScript adding .spago (#3278)	3 years ago
	<a href="#">Python.gitignore</a>	Update Python.gitignore	last year
	Qooxdoo.gitignore	Add gitignore for qooxdoo apps	13 years ago
	Qt.gitignore	Remove trailing whitespace	2 years ago
	R.gitignore	Merge pull request #3792 from jl5000/patch-1	2 years ago
	README.md	Merge pull request #3854 from AnilSeervi/patch-1	2 years ago
	ROS.gitignore	Added ignore for files created by <code>catkin_make_isolated</code>	6 years ago
	Racket.gitignore	Update Racket.gitignore	2 years ago
	Rails.gitignore	Ignore Rails .env according recommendations	2 years ago
	Raku.gitignore	Changes the name of Perl 6 to Raku (#3312)	3 years ago
	RhodesRhomobile.gitignore	Add Rhodes mobile application framework gitignore	13 years ago
	Ruby.gitignore	Ruby: ignore RuboCop remote inherited config files (#3197)	4 years ago

```
1  # Byte-compiled / optimized / DLL files
2  __pycache__/
3  *.py[cod]
4  *$py.class
5
6  # C extensions
7  *.so
8
9  # Distribution / packaging
10 .Python
11 build/
12 develop-eggs/
13 dist/
14 downloads/
15 eggs/
16 .eggs/
17 lib/
18 lib64/
19 parts/
20 sdist/
21 var/
22 wheels/
23 share/python-wheels/
24 *.egg-info/
25 .installed.cfg
26 *.egg
27 MANIFEST
28
```

3. Теперь создаём файл «README.md», где вносим информацию о своей группе и ФИО. Сохраняем набранный текст через кнопку «Commit changes».



4. В окне «Codespace» выбираем опцию «Create codespace on main». Откроется терминал, куда мы введём команду «git clone», чтобы клонировать свой репозиторий. После этого организуем репозиторий в соответствие с моделью ветвления Git-flow. Для этого введём в терминал команды: «git checkout –b develop» для создания ветки разработки; «git branch feature\_branch» для создания ветки функций; «git branch release/1.0.0» для создания ветки релиза; «git checkout main» и «git branch hotfix» для создания веток hotfix. Создаём файл .pre-commit-config.yaml и environment.yml.

```

● @RomanGorchakov →/workspaces/Py18 (main) $ git clone https://github.com/RomanGorchakov/Py18.git ClonedPy4.git
Cloning into 'ClonedPy4.git'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 9 (delta 1), reused 0 (delta 0), pack-reused 0
Receiving objects: 100% (9/9), 4.81 KiB | 4.81 MiB/s, done.
Resolving deltas: 100% (1/1), done.
● @RomanGorchakov →/workspaces/Py18 (main) $ git checkout -b develop
Switched to a new branch 'develop'
● @RomanGorchakov →/workspaces/Py18 (develop) $ git branch feature_branch
● @RomanGorchakov →/workspaces/Py18 (develop) $ git branch release/1.0.0
● @RomanGorchakov →/workspaces/Py18 (develop) $ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
● @RomanGorchakov →/workspaces/Py18 (main) $ git branch hotfix
● @RomanGorchakov →/workspaces/Py18 (main) $ git checkout develop
Switched to branch 'develop'
○ @RomanGorchakov →/workspaces/Py18 (develop) $ █

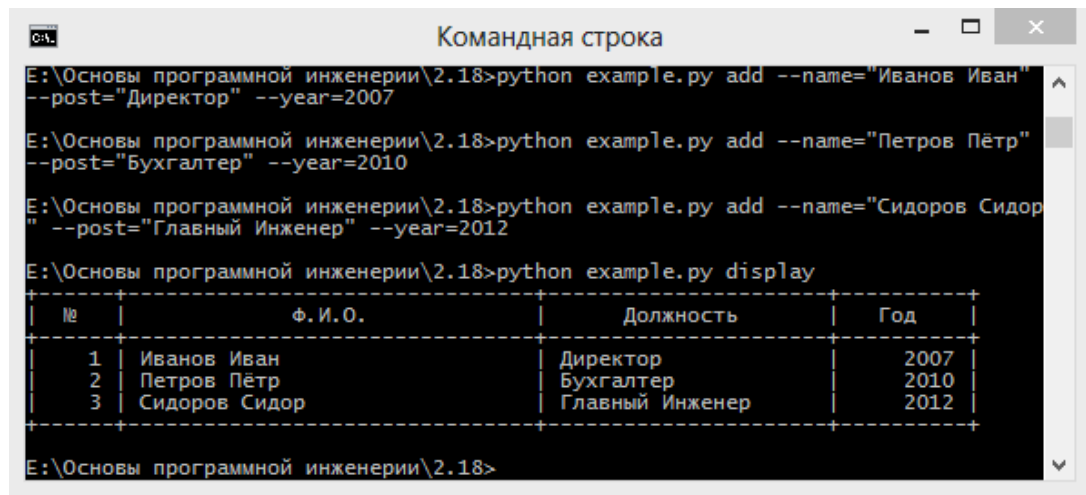
```

```

Collecting nodeenv<=0.11.1 (from pre-commit)
  Downloading nodeenv-1.8.0-py2.py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: pyyaml<=5.1 in /home/codespace/.local/lib/python3.10/site-packages (from pre-commit) (6.0.1)
Collecting virtualenv<=20.10.0 (from pre-commit)
  Downloading virtualenv-20.26.2-py3-none-any.whl.metadata (4.4 kB)
Requirement already satisfied: setuptools in /usr/local/python/3.10.13/lib/python3.10/site-packages (from nodeenv<=0.11.1->pre-commit) (68.2.2)
Collecting distlib<1,>=0.3.7 (from virtualenv<=20.10.0->pre-commit)
  Downloading distlib-0.3.8-py2.py3-none-any.whl.metadata (5.1 kB)
Requirement already satisfied: filelock<4,>=3.12.2 in /home/codespace/.local/lib/python3.10/site-packages (from virtualenv<=20.10.0->pre-commit) (3.13.3)
Requirement already satisfied: platformdirs<5,>=3.9.1 in /home/codespace/.local/lib/python3.10/site-packages (from virtualenv<=20.10.0->pre-commit) (4.2.0)
Downloading pre_commit-3.7.1-py2.py3-none-any.whl (204 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 204.3/204.3 kB 4.3 MB/s eta 0:00:00
Downloading cfgv-3.4.0-py2.py3-none-any.whl (7.2 kB)
Downloading identify-2.5.36-py2.py3-none-any.whl (98 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 99.0/99.0 kB 3.2 MB/s eta 0:00:00
Downloading nodeenv-1.8.0-py2.py3-none-any.whl (22 kB)
Downloading virtualenv-20.26.2-py3-none-any.whl (3.9 MB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.9/3.9 MB 49.2 MB/s eta 0:00:00
Downloading distlib-0.3.8-py2.py3-none-any.whl (468 kB)
  ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 468.9/468.9 kB 13.0 MB/s eta 0:00:00
Installing collected packages: distlib, virtualenv, nodeenv, identify, cfgv, pre-commit
Successfully installed cfgv-3.4.0 distlib-0.3.8 identify-2.5.36 nodeenv-1.8.0 pre-commit-3.7.1 virtualenv-20.26.2
● @RomanGorchakov →/workspaces/Py18 (develop) $ pre-commit sample-config > .pre-commit-config.yaml
● @RomanGorchakov →/workspaces/Py18 (develop) $ conda env export > environment.yml
○ @RomanGorchakov →/workspaces/Py18 (develop) $ █

```

5. Создаём файл «example.py», в котором нужно в котором нужно добавить возможность получения имени файла данных для примера из лабораторной работы 2.16.

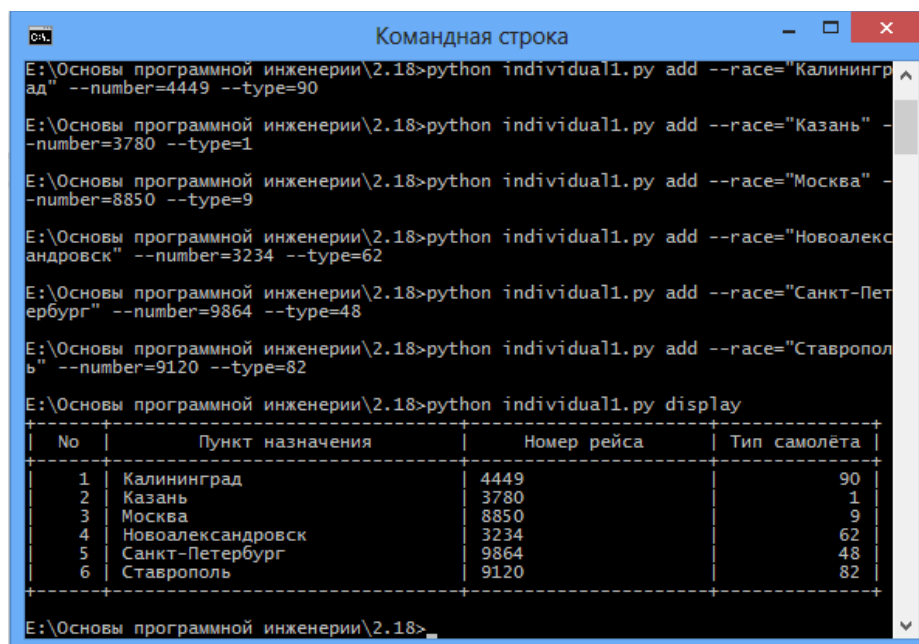


```
E:\Основы программной инженерии\2.18>python example.py add --name="Иванов Иван" --post="Директор" --year=2007
E:\Основы программной инженерии\2.18>python example.py add --name="Петров Пётр" --post="Бухгалтер" --year=2010
E:\Основы программной инженерии\2.18>python example.py add --name="Сидоров Сидор" --post="Главный Инженер" --year=2012
E:\Основы программной инженерии\2.18>python example.py display
```

№	Ф.И.О.	Должность	Год
1	Иванов Иван	Директор	2007
2	Петров Пётр	Бухгалтер	2010
3	Сидоров Сидор	Главный Инженер	2012

```
E:\Основы программной инженерии\2.18>
```

6. Создаём файл «individual1.py», в котором нужно добавить возможность получения имени файла данных, используя соответствующую переменную окружения. Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по возрастанию номера рейса; вывод на экран номеров рейсов и типов самолетов, вылетающих в пункт назначения, название которого совпало с названием, введенным с клавиатуры; если таких рейсов нет, выдать на дисплей соответствующее сообщение. Необходимо также проследить за тем, чтобы файлы генерируемый этой программой не попадали в репозиторий лабораторной работы.

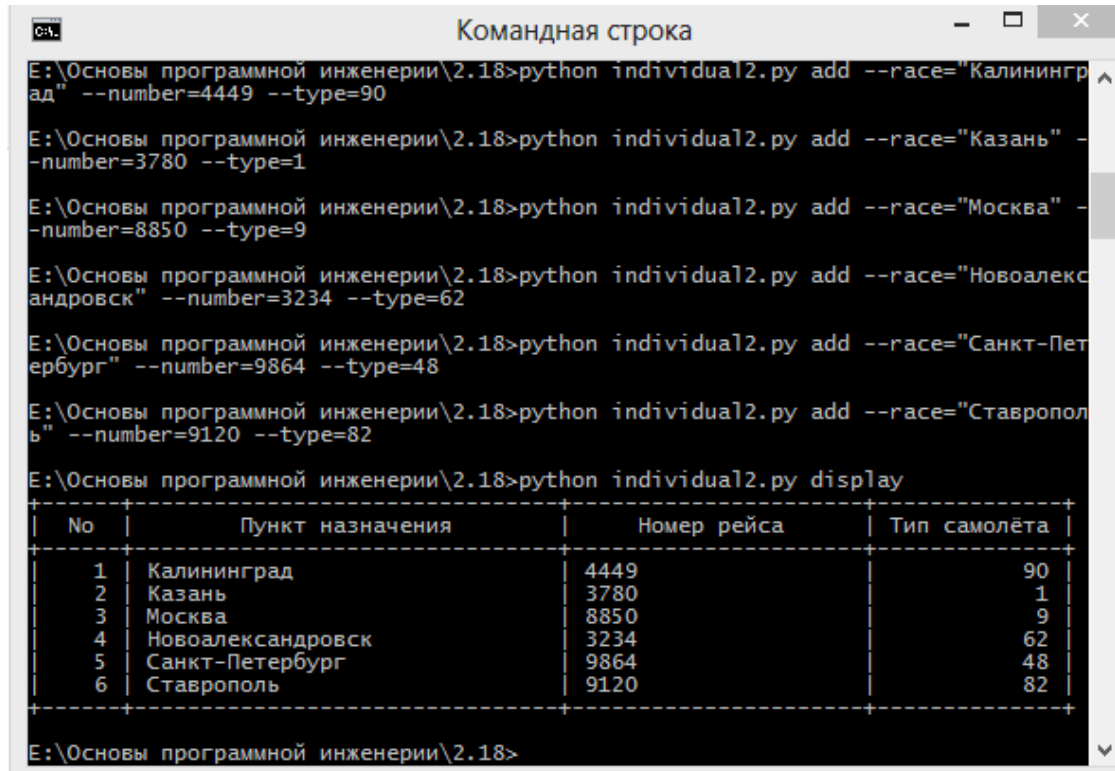


```
E:\Основы программной инженерии\2.18>python individual1.py add --race="Калининград" --number=4449 --type=90
E:\Основы программной инженерии\2.18>python individual1.py add --race="Казань" --number=3780 --type=1
E:\Основы программной инженерии\2.18>python individual1.py add --race="Москва" --number=8850 --type=9
E:\Основы программной инженерии\2.18>python individual1.py add --race="Новоалександровск" --number=3234 --type=62
E:\Основы программной инженерии\2.18>python individual1.py add --race="Санкт-Петербург" --number=9864 --type=48
E:\Основы программной инженерии\2.18>python individual1.py add --race="Ставрополь" --number=9120 --type=82
E:\Основы программной инженерии\2.18>python individual1.py display
```

No	Пункт назначения	Номер рейса	Тип самолёта
1	Калининград	4449	90
2	Казань	3780	1
3	Москва	8850	9
4	Новоалександровск	3234	62
5	Санкт-Петербург	9864	48
6	Ставрополь	9120	82

```
E:\Основы программной инженерии\2.18>
```

7. Создаём файл «individual2.py», в котором нужно модифицировать программу таким образом, чтобы значения необходимых переменных окружения считывались из файла .env.



```
E:\Основы программной инженерии\2.18>python individual2.py add --race="Калининград" --number=4449 --type=90

E:\Основы программной инженерии\2.18>python individual2.py add --race="Казань" --number=3780 --type=1

E:\Основы программной инженерии\2.18>python individual2.py add --race="Москва" --number=8850 --type=9

E:\Основы программной инженерии\2.18>python individual2.py add --race="Новоалександровск" --number=3234 --type=62

E:\Основы программной инженерии\2.18>python individual2.py add --race="Санкт-Петербург" --number=9864 --type=48

E:\Основы программной инженерии\2.18>python individual2.py add --race="Ставрополь" --number=9120 --type=82

E:\Основы программной инженерии\2.18>python individual2.py display
```

No	Пункт назначения	Номер рейса	Тип самолёта
1	Калининград	4449	90
2	Казань	3780	1
3	Москва	8850	9
4	Новоалександровск	3234	62
5	Санкт-Петербург	9864	48
6	Ставрополь	9120	82

```
E:\Основы программной инженерии\2.18>
```

8. Выполняем коммит файлов в репозиторий Git в ветку разработки, сливаем её с веткой main и отправляем изменения на сервер GitHub.

```
rd/packet.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\321\213\320\265\320\267\320\260\320\264\320\260\320\275\320\270\321\217\320\227\320\260\320\264\320\260\320\275\320\270\320\265\2/packet_ha
rd/save_plane.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\321\213\320\265\320\267\320\260\320\264\320\260\320\275\320\270\321\217\320\227\320\260\320\264\320\260\320\275\320\270\320\265\2/packet_ha
rd/show_plane.py"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\321\213\320\265\320\267\320\260\320\264\320\260\320\275\320\270\321\217\320\227\320\260\320\264\320\260\320\275\320\270\320\265\2/packet_ha
rd/show_plane.py.bak"
create mode 100644 "code/\320\230\320\275\320\264\320\270\320\262\320\270\320\264\321\203\320\260\320\273\321\214\320\275\321\213\320\265\320\267\320\260\320\264\320\260\320\275\320\270\321\217\320\227\320\260\320\264\320\260\320\275\320\270\320\265\2/schedule-
hard.json"
create mode 100644 "code/\320\237\321\200\320\270\320\274\320\265\321\200\data.json"
create mode 100644 "code/\320\237\321\200\320\270\320\274\320\265\321\200/example.py"
create mode 100644 environment.yml
@RomanGorchakov →/workspaces/Py18 (main) $ git push -u
Enumerating objects: 29, done.
Counting objects: 100% (29/29), done.
Delta compression using up to 2 threads
Compressing objects: 100% (28/28), done.
Writing objects: 100% (28/28), 6.56 KiB | 1.09 MiB/s, done.
Total 28 (delta 9), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (9/9), done.
To https://github.com/RomanGorchakov/Py18
3f9acb8..fa77b64 main -> main
branch 'main' set up to track 'origin/main'.
@RomanGorchakov →/workspaces/Py18 (main) $
```

main 1 Branch 0 Tags

Go to file t Add file <> Code

RomanGorchakov docs ac96cf9 · now 6 Commits

code	Code and doc	2 minutes ago
doc	docs	now
.gitignore	Create .gitignore	2 hours ago
.pre-commit-config.yaml	Environment variables	3 minutes ago
LICENSE	Create LICENSE	2 hours ago
README.md	Create README.md	2 hours ago
environment.yml	Environment variables	3 minutes ago

README MIT license

## Лабораторная работа 2.18. Работа с переменными окружения в Python3.

### Контрольные вопросы

#### 1. Каково назначение переменных окружения?

Переменная среды (переменная окружения) – это короткая ссылка на какой-либо объект в системе. С помощью таких сокращений, например, можно создавать универсальные пути для приложений, которые будут работать на любых ПК, независимо от имен пользователей и других параметров.

#### 2. Какая информация может храниться в переменных окружения?

Переменные окружения могут быть использованы для хранения информации, которую используют различные приложения, и для настройки системных параметров.

#### 3. Как получить доступ к переменным окружения в ОС Windows?

Получить информацию о существующих переменных можно в свойствах системы. Для этого кликаем по ярлыку Компьютера на рабочем столе правой кнопкой мыши и выбираем соответствующий пункт. Переходим в «Дополнительные параметры». В открывшемся окне с вкладкой «Дополнительно»



нажимаем кнопку, указанную на скриншоте ниже. Здесь мы видим два блока. Первый содержит пользовательские переменные, а второй системные.

#### 4. Каково назначение переменных PATH и PATHNEXT?

PATH позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. PATHNEXT, в свою очередь, дает возможность не указывать даже расширение файла, если оно прописано в ее значениях.

#### 5. Как создать или изменить переменную окружения в Windows?

Создание переменной окружения:

- нажимаем кнопку Создать. Сделать это можно как в пользовательском разделе, так и в системном;
- вводим имя, например, desktop. Обратите внимание на то, чтобы такое название еще не было использовано (просмотрите списки);
- в поле Значение указываем путь до папки Рабочий стол;
- нажимаем ОК. Повторяем это действие во всех открытых окнах;
- перезапускаем Проводник и консоль или целиком систему;
- готово, новая переменная создана, увидеть ее можно в соответствующем списке.

Изменение переменной окружения:

- нажимаем кнопку Изменить;
- вводим имя, например, desktop;
- в поле Значение указываем путь до папки Рабочий стол;
- нажимаем ОК. Повторяем это действие во всех открытых окнах;
- перезапускаем Проводник и консоль или целиком систему;
- готово, переменная изменена.

#### 6. Что представляют собой переменные окружения в ОС Linux?

Переменные окружения в Linux представляют собой набор именованных значений, используемых другими приложениями. Переменные окружения применяются для настройки поведения приложений и работы самой системы. Например, переменная окружения может хранить информацию о путях к

исполняемым файлам, заданном по умолчанию текстовом редакторе, браузере, языковых параметрах (локали) системы или настройках раскладки клавиатуры.

#### 7. В чем отличие переменных окружения от переменных оболочки?

Переменные окружения (или «переменные среды») – переменные, доступные в масштабах всей системы и наследуемые всеми дочерними процессами и оболочками.

Переменные оболочки – переменные, которые применяются только к текущему экземпляру оболочки. Каждая оболочка, например, `bash` или `zsh`, имеет свой собственный набор внутренних переменных.

#### 8. Как вывести значение переменной окружения в Linux?

Наиболее часто используемая команда для вывода переменных окружения – `printenv`. Если команде в качестве аргумента передать имя переменной, то будет отображено значение только этой переменной.

#### 9. Какие переменные окружения Linux Вам известны?

- `USER` – текущий пользователь.
- `PWD` – текущая директория.
- `OLDPWD` – предыдущая рабочая директория. Используется оболочкой для того, чтобы вернуться в предыдущий каталог при выполнении команды `cd -`.
- `HOME` – домашняя директория текущего пользователя.
- `SHELL` – путь к оболочке текущего пользователя (например, `bash` или `zsh`).
- `EDITOR` – заданный по умолчанию редактор. Этот редактор будет вызываться в ответ на команду `edit`.
- `LOGNAME` – имя пользователя, используемое для входа в систему.
- `PATH` – пути к каталогам, в которых будет производиться поиск вызываемых команд. При выполнении команды система будет проходить по данным каталогам в указанном порядке и выберет первый из них, в котором будет находиться исполняемый файл искомой команды.
- `LANG` – текущие настройки языка и кодировки.
- `TERM` – тип текущего эмулятора терминала.

- MAIL – место хранения почты текущего пользователя.
- LS\_COLORS – задает цвета, используемые для выделения объектов (например, различные типы файлов в выводе команды ls будут выделены разными цветами).

#### 10. Какие переменные оболочки Linux Вам известны?

- BASHOPTS – список задействованных параметров оболочки, разделенных двоеточием.
- BASH\_VERSION – версия запущенной оболочки bash.
- COLUMNS – количество столбцов, которые используются для отображения выходных данных.
- DIRSTACK – стек директорий, к которому можно применять команды pushd и popd.
- HISTFILESIZE – максимальное количество строк для файла истории команд.
- HISTSIZE – количество строк из файла истории команд, которые можно хранить в памяти.
- HOSTNAME – имя текущего хоста.
- IFS – внутренний разделитель поля в командной строке (по умолчанию используется пробел).
- PS1 – определяет внешний вид строки приглашения ввода новых команд.
- PS2 – вторичная строка приглашения.
- SHELL\_OPTS – параметры оболочки, которые можно устанавливать с помощью команды set.
- UID – идентификатор текущего пользователя.

#### 11. Как установить переменные оболочки в Linux?

Чтобы создать новую переменную оболочки с именем, например, NEW\_VAR и значением Ravesli.com, просто введите: \$ NEW\_VAR=Ravesli.com».

#### 12. Как установить переменные окружения в Linux?

Для создания переменной окружения экспортируем нашу недавно созданную переменную оболочки: `$ export NEW_VAR`.

13. Для чего необходимо делать переменные окружения Linux постоянными?

Чтобы установить постоянную переменную окружения для текущего пользователя, откройте файл `.**bashrc`: `$ sudo nano ~/.bashrc`.

Для каждой переменной, которую вы хотите сделать постоянной, добавьте в конец файла строку, используя следующий синтаксис: `export [ИМЯ_ПЕРЕМЕННОЙ]=[ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ]`.

14. Для чего используется переменная окружения PYTHONHOME?

Переменная среды PYTHONHOME изменяет расположение стандартных библиотек Python. По умолчанию библиотеки ищутся в `prefix/lib/pythonversion` и `exec_prefix/lib/pythonversion`, где `prefix` и `exec_prefix` – каталоги, зависящие от установки, оба каталога по умолчанию - `/usr/local`.

15. Для чего используется переменная окружения PYTHONPATH?

Переменная среды PYTHONPATH изменяет путь поиска по умолчанию для файлов модуля. Формат такой же, как для оболочки PATH: один или несколько путей к каталогам, разделенных `os.pathsep` (например, двоеточие в Unix или точка с запятой в Windows). Несуществующие каталоги игнорируются.

16. Какие ещё переменные окружения используются для управления работой интерпретатора Python?

PYTHONSTARTUP, PYTHONOPTIMIZE, PYTHONBREAKPOINT,  
PYTHONDEBUG, PYTHONINSPECT, PYTHONUNBUFFERED,  
PYTHONVERBOSE, PYTHONCASEOK, PYTHONDONTWRITEBYTECODE,  
PYTHONPYCACHEPREFIX, PYTHONHASHSEED, PYTHONIOENCODING,  
PYTHONNOUSERSITE, PYTHONUSERBASE, PYTHONWARNINGS,  
PYTHONFAULTHANDLER, PYTHONTRACEMALLOC, PYTHONUTF8,  
PYTHONPROFILEIMPORTTIME, PYTHONASYNCIODEBUG, PYTHONMALLOC,  
PYTHONMALLOCSTATS, PYTHONLEGACYWINDOWSFSENCODING,  
PYTHONLEGACYWINDOWSSSTDIO, PYTHONCOERCECLOCALE,

PYTHONDEVMODE,

PYTHONWARNDEFAULTENCODING,

PYTHONTHREADDEBUG, PYTHONDUMPREFS.

17. Как осуществляется чтение переменных окружения в программах на языке программирования Python?

Считываем одну или все переменные окружения, проверяем, присвоено ли значение переменной окружения, проверяем переменную на истинность, присваиваем значение переменной окружения.

18. Как проверить, установлено или нет значение переменной окружения в программах на языке программирования Python?

Бесконечный цикл while непрерывно принимает от пользователя имена переменных и проверяет их значения до тех пор, пока пользователь не введёт имя переменной, которой не присвоено значение. Если пользователь вводит имя переменной окружения, которой присвоено значение, это значение выводится, если же нет – выводится соответствующее сообщение и процесс останавливается.

19. Как присвоить значение переменной окружения в программах на языке программирования Python?

Для присвоения значения любой переменной среды используется функция setdefault().