

# Лабораторная работа 2.18. Работа с переменными окружения в Python3

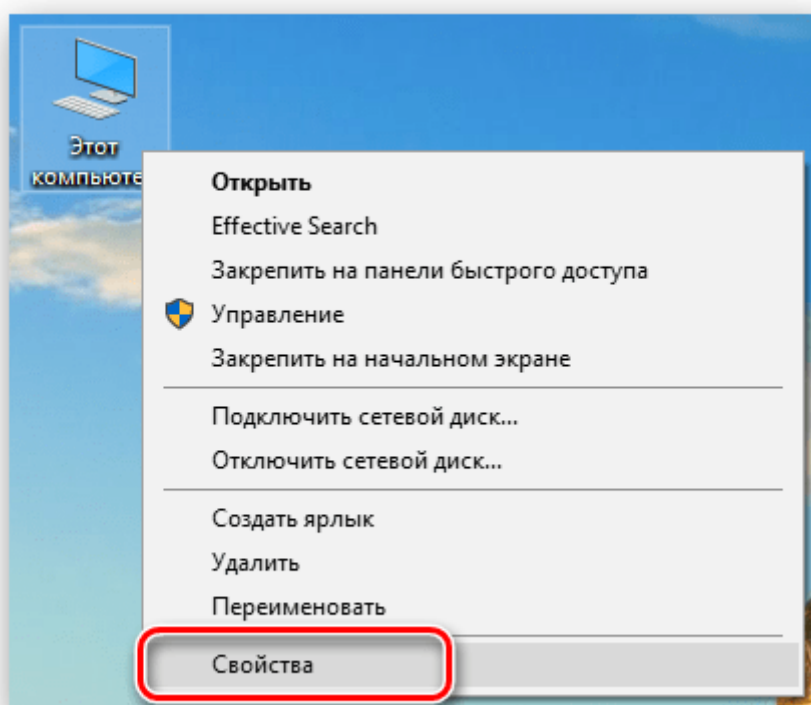
**Цель работы:** приобретение навыков по работе с переменными окружения с помощью языка программирования Python версии 3.x.

## Ход работы

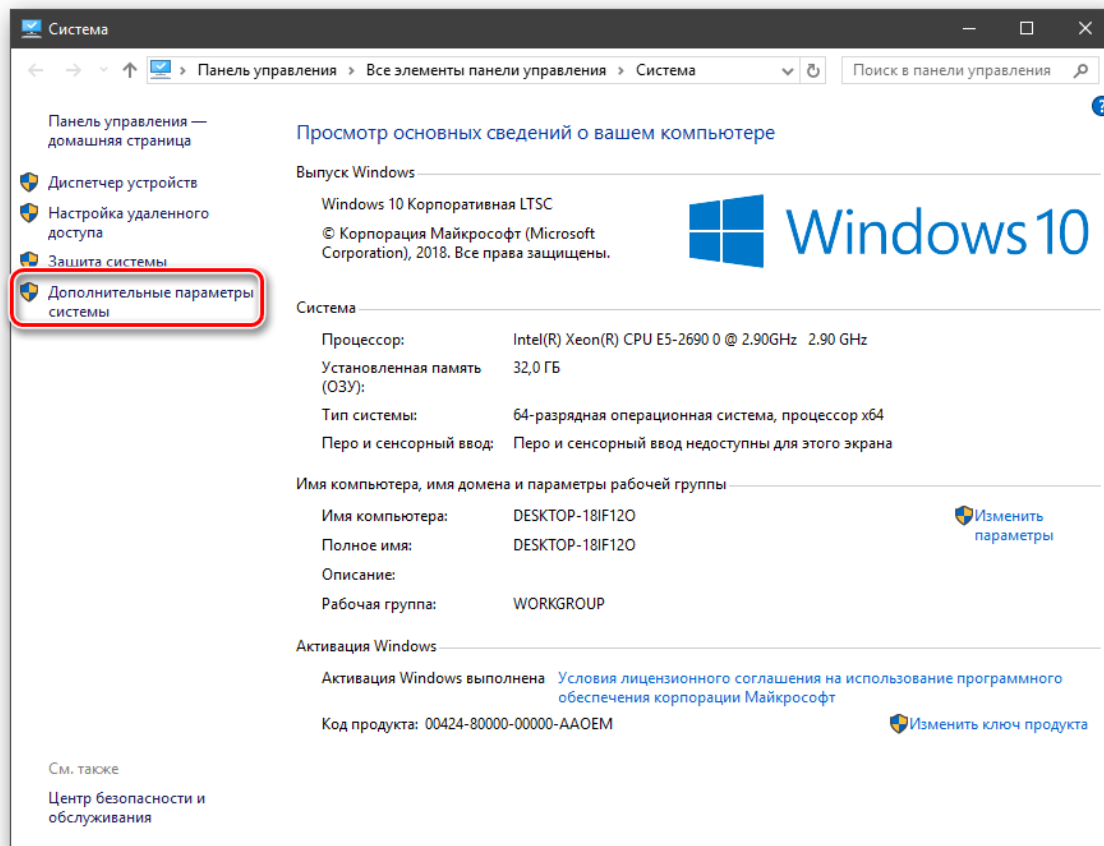
Переменная среды (переменная окружения) – это короткая ссылка на какой-либо объект в системе. С помощью таких сокращений, например, можно создавать универсальные пути для приложений, которые будут работать на любых ПК, независимо от имен пользователей и других параметров.

## Переменные среды Windows

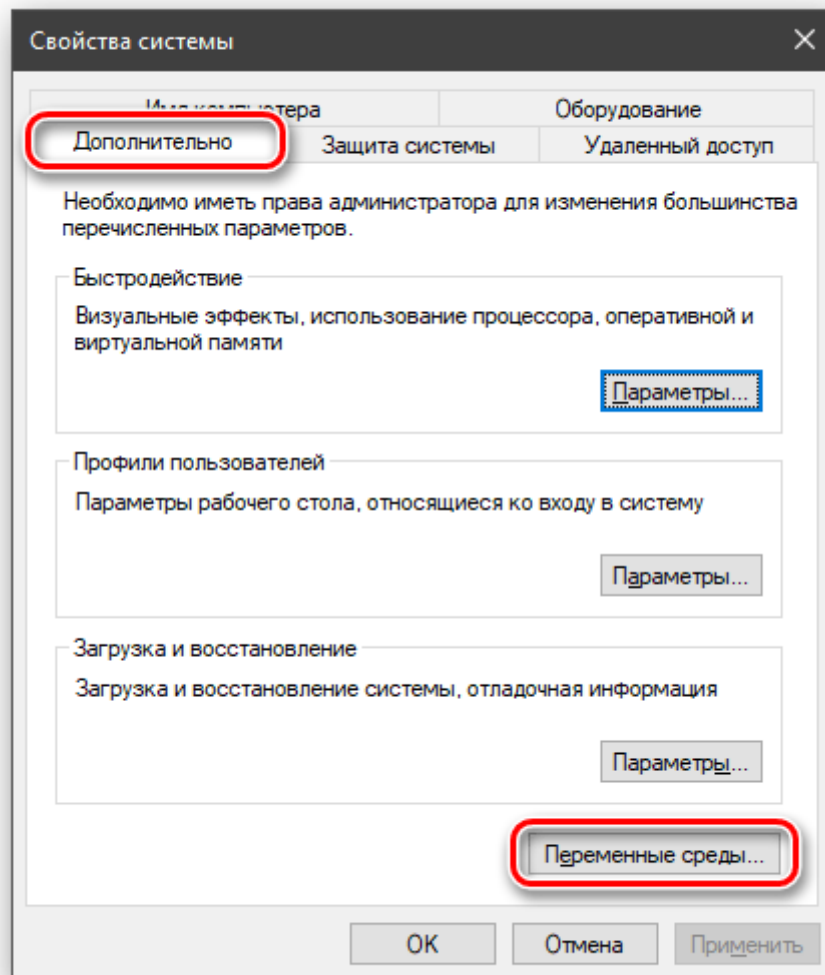
Получить информацию о существующих переменных можно в свойствах системы. Для этого кликаем по ярлыку Компьютера на рабочем столе правой кнопкой мыши и выбираем соответствующий пункт.



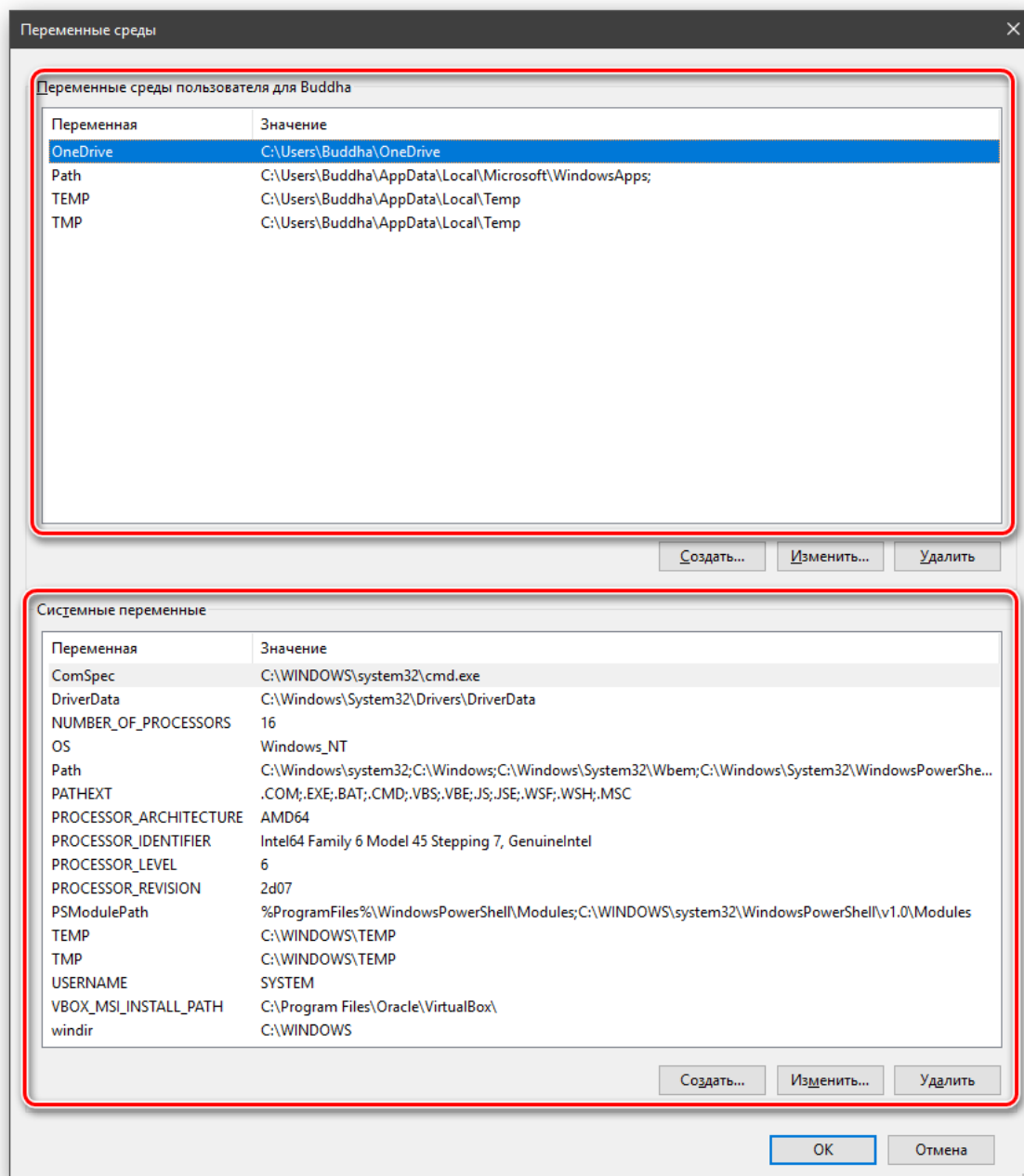
Переходим в «Дополнительные параметры».



В открывшемся окне с вкладкой «Дополнительно» нажимаем кнопку, указанную на скриншоте ниже.

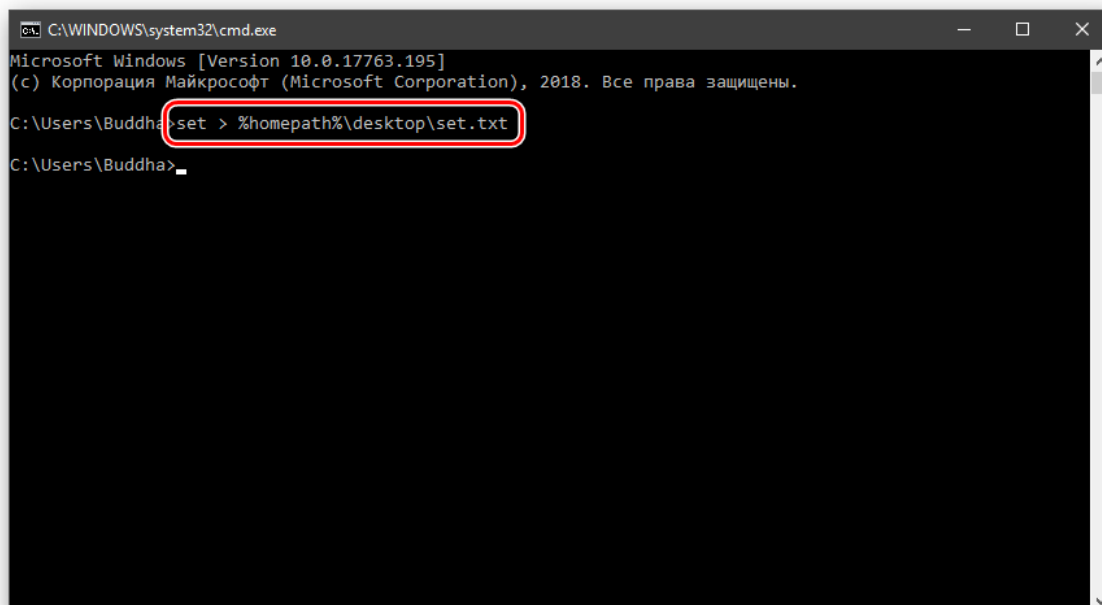


Здесь мы видим два блока. Первый содержит пользовательские переменные, а второй системные.



Если требуется просмотреть весь перечень, запускаем «Командную строку» от имени администратора и выполняем команду (вводим и нажимаем *ENTER*).

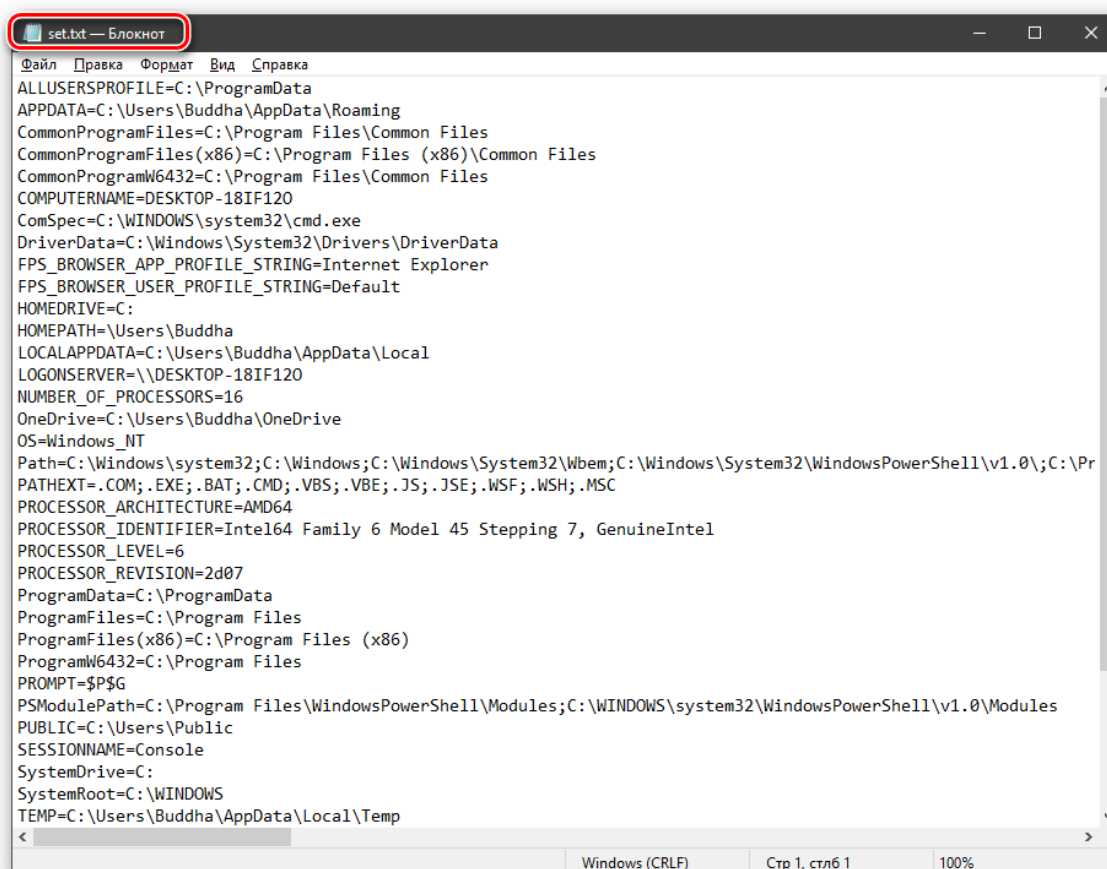
```
set > %homepath%\desktop\set.txt
```



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.195]
(c) Корпорация Майкрософт (Microsoft Corporation), 2018. Все права защищены.

C:\Users\Buddha>set > %homepath%\desktop\set.txt
C:\Users\Buddha>
```

На рабочем столе появится файл с названием «*set.txt*», в котором будут указаны все переменные окружения, имеющиеся в системе.



```
set.txt — Блокнот
Файл Правка Формат Вид Справка
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\Buddha\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
COMPUTERNAME=DESKTOP-18IF120
ComSpec=C:\WINDOWS\system32\cmd.exe
DriverData=C:\Windows\System32\Drivers\DriverData
FPS_BROWSER_APP_PROFILE_STRING=Internet Explorer
FPS_BROWSER_USER_PROFILE_STRING=Default
HOMEDRIVE=C:
HOMEPATH=\Users\Buddha
LOCALAPPDATA=C:\Users\Buddha\AppData\Local
LOGONSERVER=\\DESKTOP-18IF120
NUMBER_OF_PROCESSORS=16
OneDrive=C:\Users\Buddha\OneDrive
OS=Windows_NT
Path=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Pr
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE=AMD64
PROCESSOR_IDENTIFIER=Intel64 Family 6 Model 45 Stepping 7, GenuineIntel
PROCESSOR_LEVEL=6
PROCESSOR_REVISION=2d07
ProgramData=C:\ProgramData
ProgramFiles=C:\Program Files
ProgramFiles(x86)=C:\Program Files (x86)
ProgramW6432=C:\Program Files
PROMPT=$P$G
PSModulePath=C:\Program Files\WindowsPowerShell\Modules;C:\WINDOWS\system32\WindowsPowerShell\v1.0\Modules
PUBLIC=C:\Users\Public
SESSIONNAME=Console
SystemDrive=C:
SystemRoot=C:\WINDOWS
TEMP=C:\Users\Buddha\AppData\Local\Temp
<
Windows (CRLF) Стр 1, столб 1 100%
```

Все их можно использовать в консоли или скриптах для запуска программ или поиска объектов, заключив имя в знаки процента. Например, в команде выше вместо пути

```
C:\Users\Имя_пользователя
```

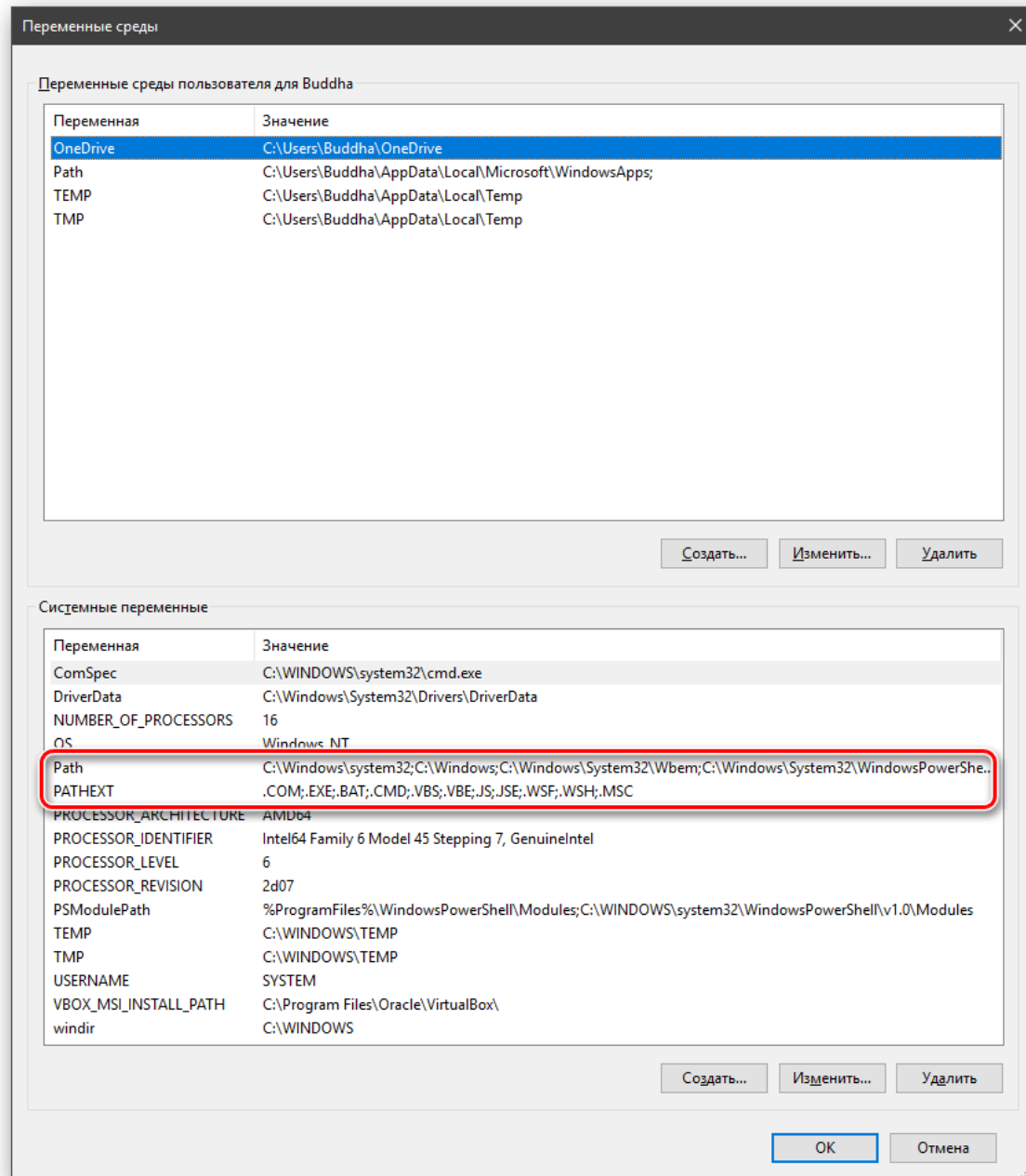
мы использовали

%homepath%

Примечание: регистр при написании переменных не важен. Path=path=PATH

## Переменные PATH и PATHEXT

Если с обычными переменными все понятно (одна ссылка – одно значение), то эти две стоят особняком. При детальном рассмотрении видно, что они ссылаются сразу на несколько объектов. Давайте разберемся, как это работает.

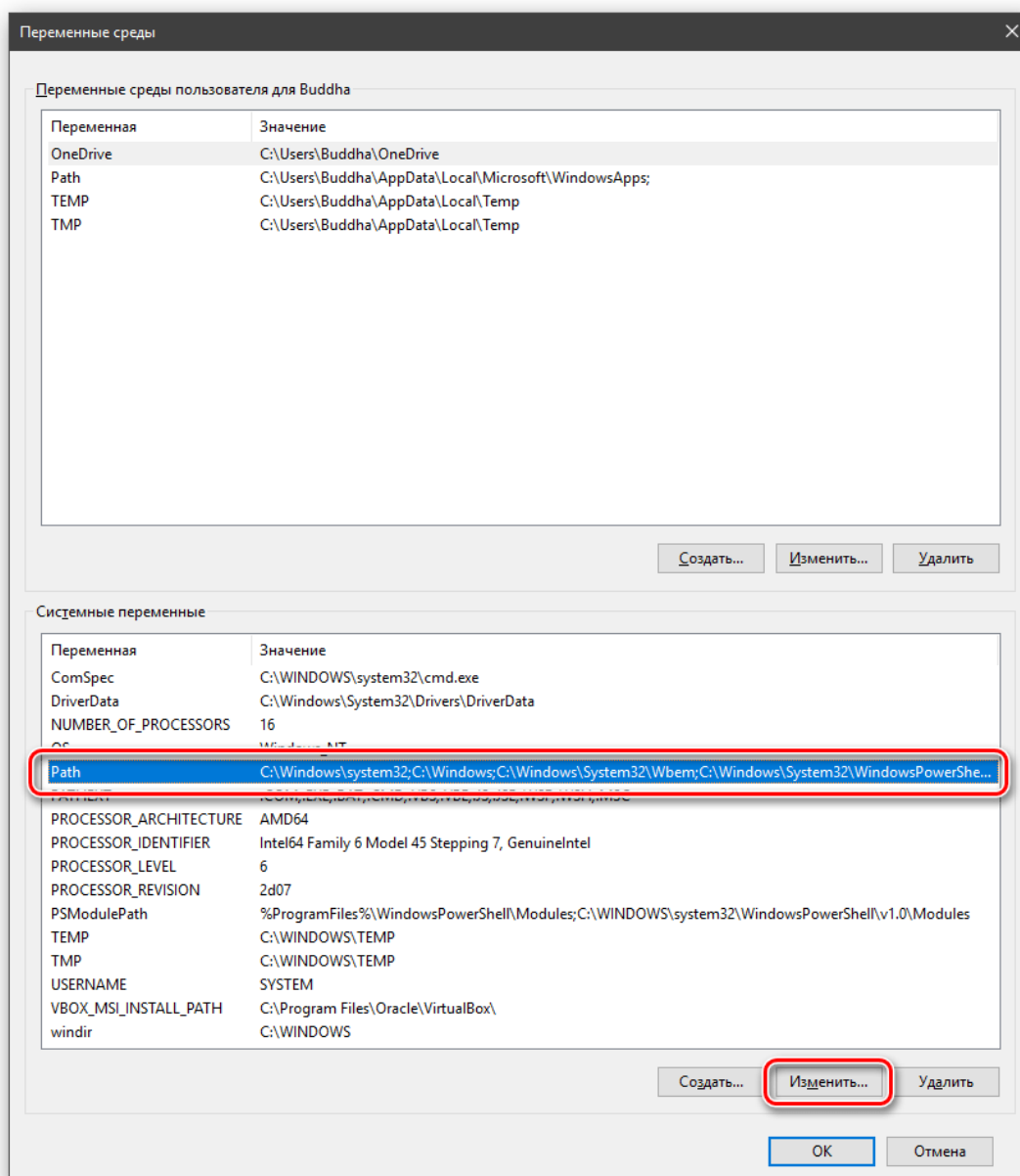


«PATH» позволяет запускать исполняемые файлы и скрипты, «лежащие» в определенных каталогах, без указания их точного местоположения. Например, если ввести в «Командную строку»

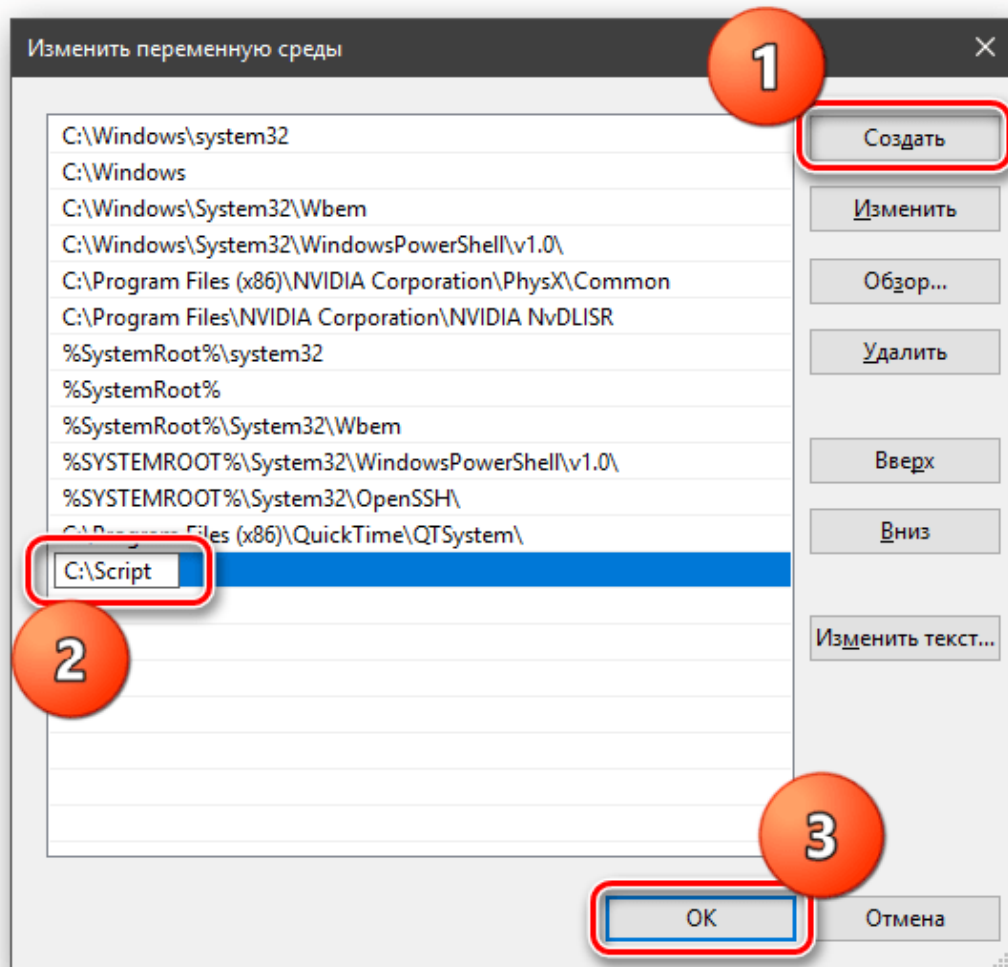
explorer.exe

система осуществит поиск по папкам, указанным в значении переменной, найдет и запустит соответствующую программу. Этим можно воспользоваться в своих целях двумя способами:

- Поместить необходимый файл в одну из указанных директорий. Полный список можно получить, выделив переменную и нажав **Изменить....**

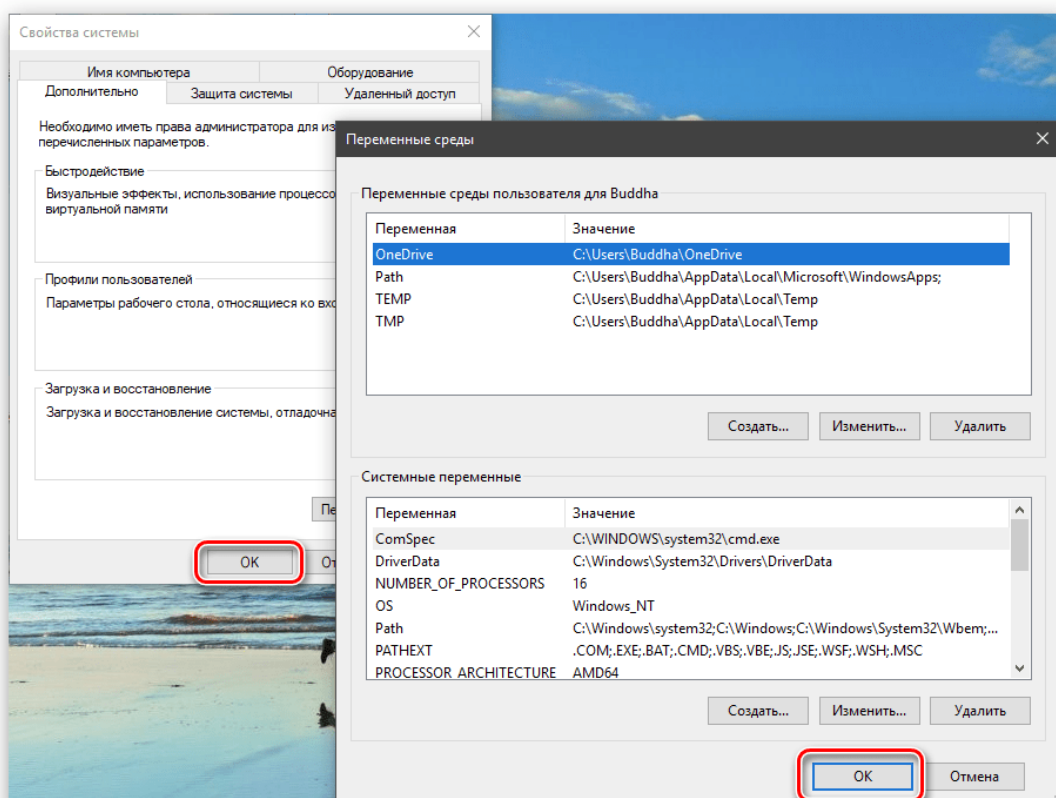


- Создать свою папку в любом месте и прописать путь к ней. Для этого (после создания директории на диске) жмем **Создать**, вводим адрес и **ОК**.



**%SYSTEMROOT%** определяет путь до папки «**Windows**» независимо от буквы диска.

Затем нажимаем **OK** в окнах **Переменные среды** и **Свойства системы**.

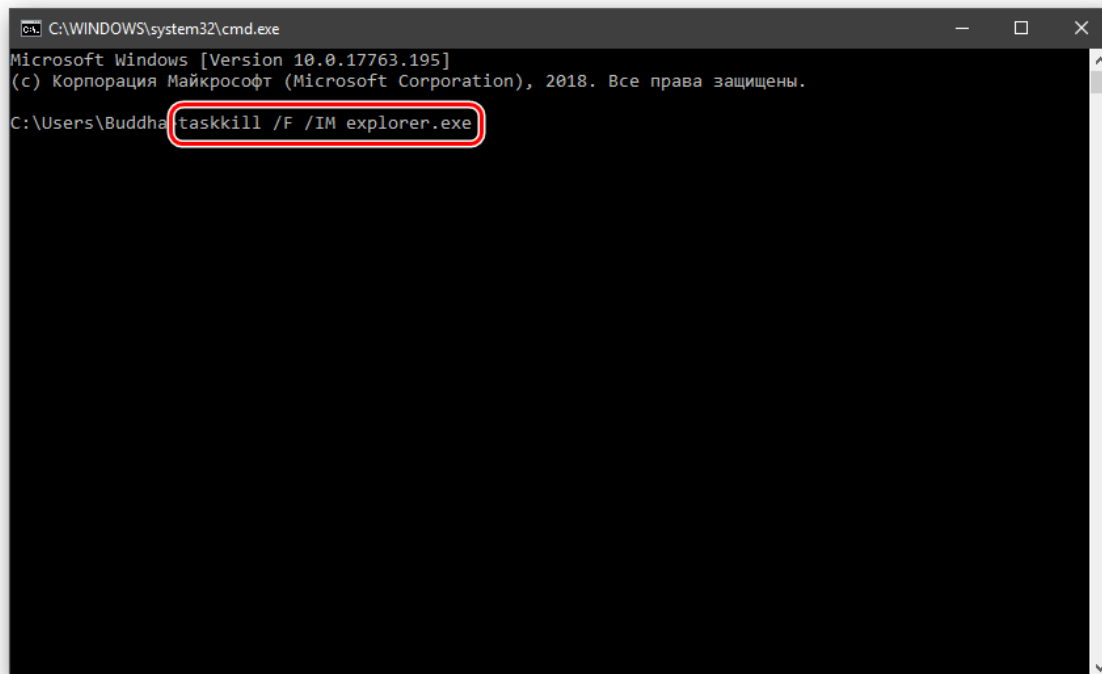




Для применения настроек, возможно, придется перезапустить **Проводник**. Сделать это быстро можно так:

Открываем «Командную строку» и пишем команду

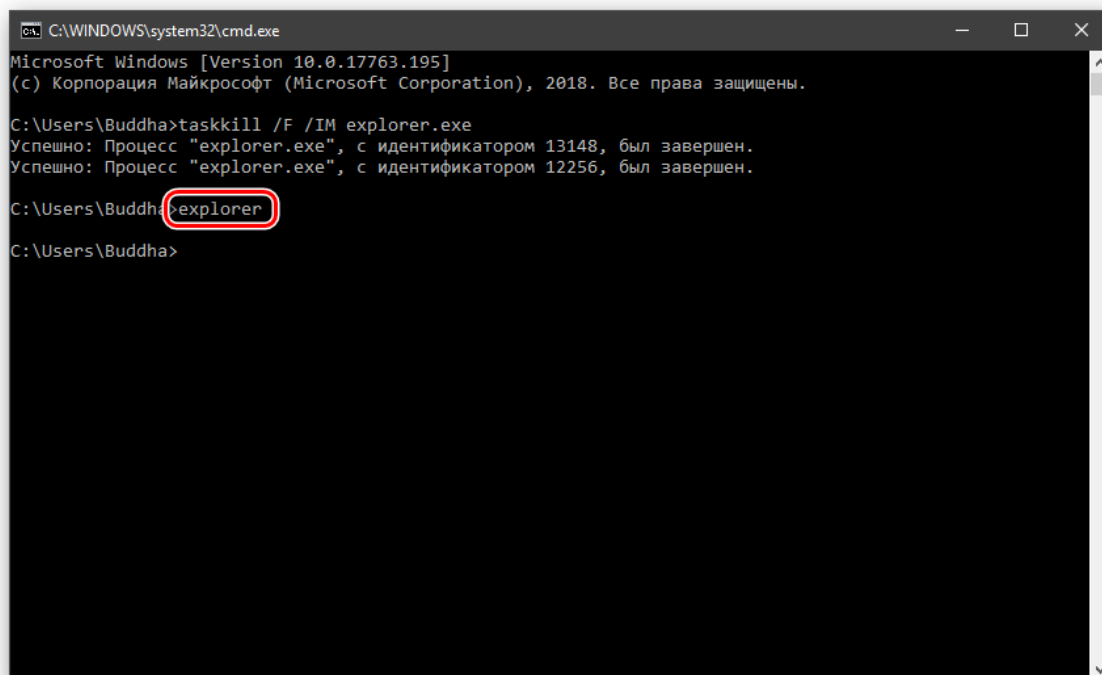
```
taskkill /F /IM explorer.exe
```



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.195]
(c) Корпорация Майкрософт (Microsoft Corporation), 2018. Все права защищены.
C:\Users\Buddha>taskkill /F /IM explorer.exe
```

Все папки и «Панель задач» исчезнут. Далее снова запускаем «Проводник».

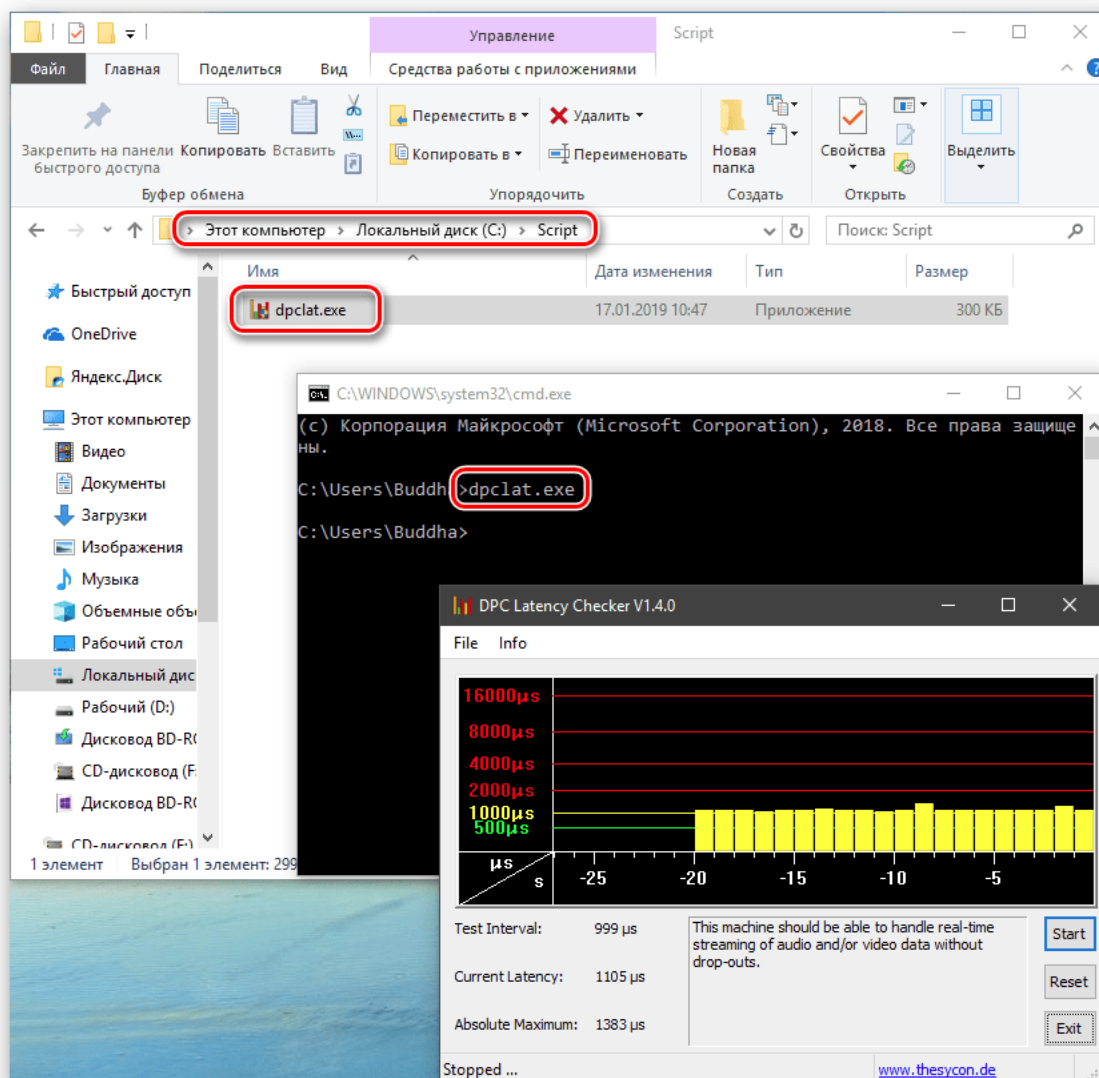
```
explorer
```



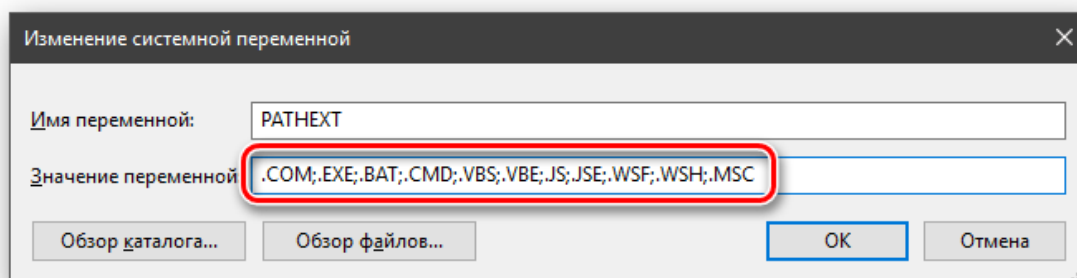
```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17763.195]
(c) Корпорация Майкрософт (Microsoft Corporation), 2018. Все права защищены.
C:\Users\Buddha>taskkill /F /IM explorer.exe
Успешно: Процесс "explorer.exe", с идентификатором 13148, был завершен.
Успешно: Процесс "explorer.exe", с идентификатором 12256, был завершен.
C:\Users\Buddha>explorer
C:\Users\Buddha>
```

Еще один момент: если вы работали с «Командной строкой», ее также следует перезапустить, то есть консоль не будет «знать», что настройки изменились. Это же касается и фреймворков, в которых вы отлаживаете свой код. Также можно перезагрузить компьютер или выйти и снова зайти в систему.

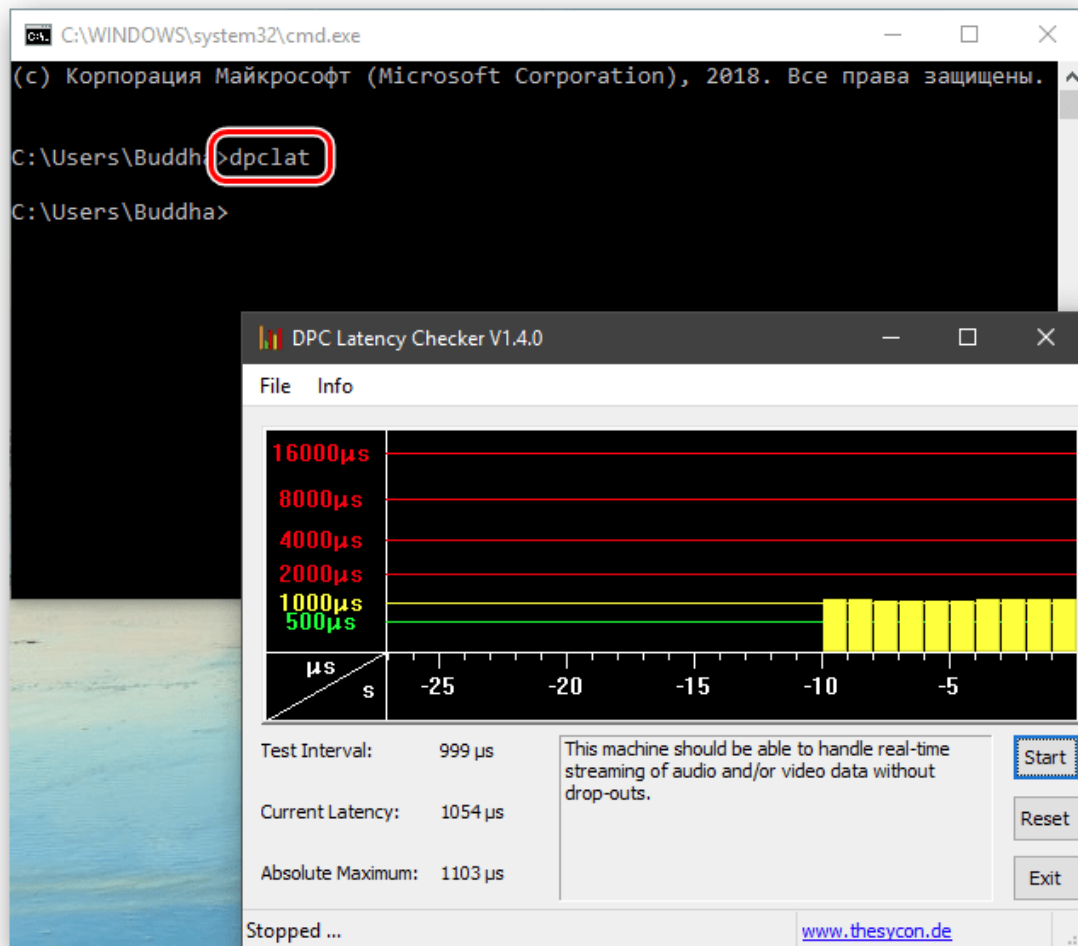
Теперь все файлы, помещенные в «C:\Script» можно будет открывать (запускать), введя только их название.



**PATHEXT**, в свою очередь, дает возможность не указывать даже расширение файла, если оно прописано в ее значениях.

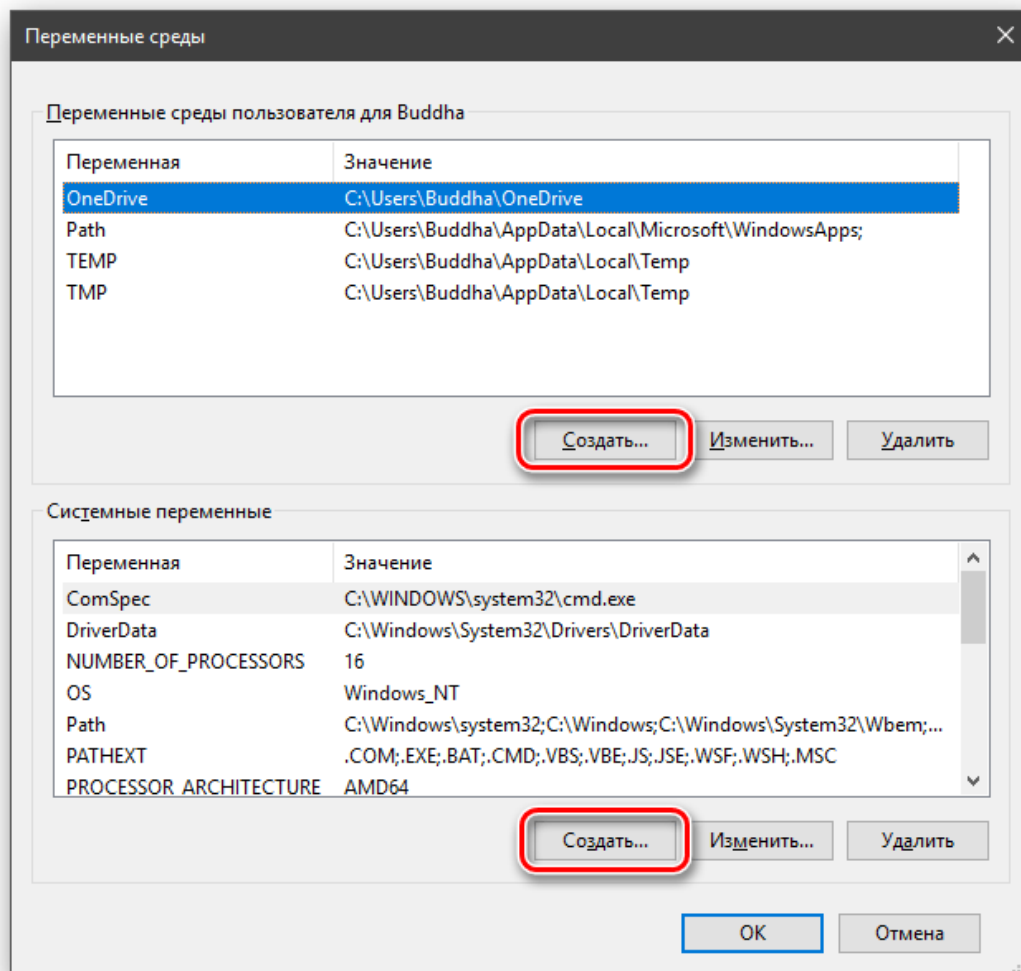


Принцип работы следующий: система перебирает расширения по очереди, пока не будет найден соответствующий объект, причем делает это в директориях, указанных в **PATH**.

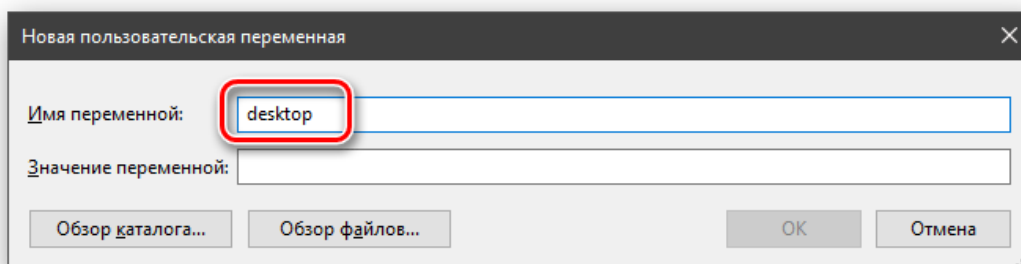


## Создание переменных среды

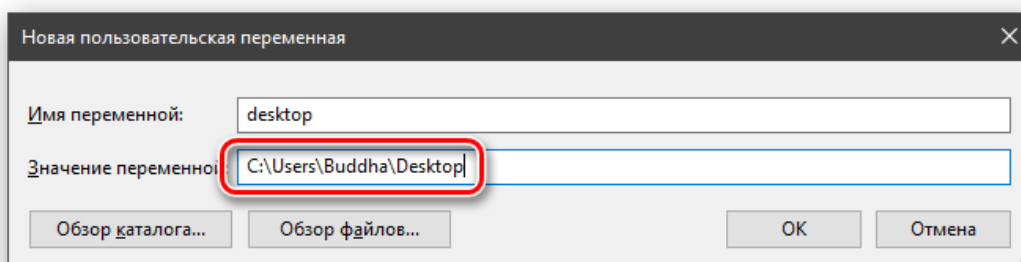
1. Нажимаем кнопку **Создать**. Сделать это можно как в пользовательском разделе, так и в системном.



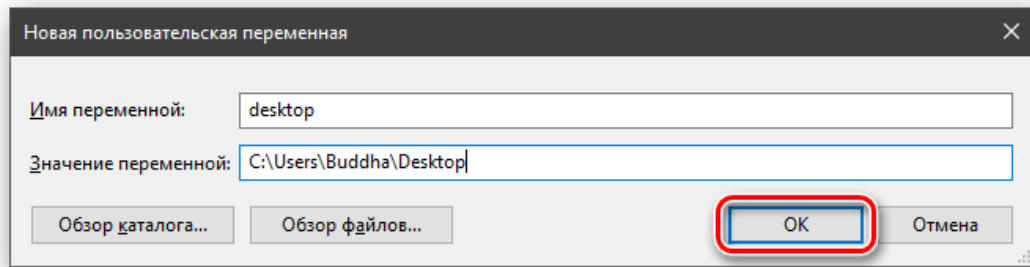
- Вводим имя, например, **desktop**. Обратите внимание на то, чтобы такое название еще не было использовано (просмотрите списки).



- В поле **Значение** указываем путь до папки **Рабочий стол**:  
C:\Users\Имя\_пользователя\Desktop

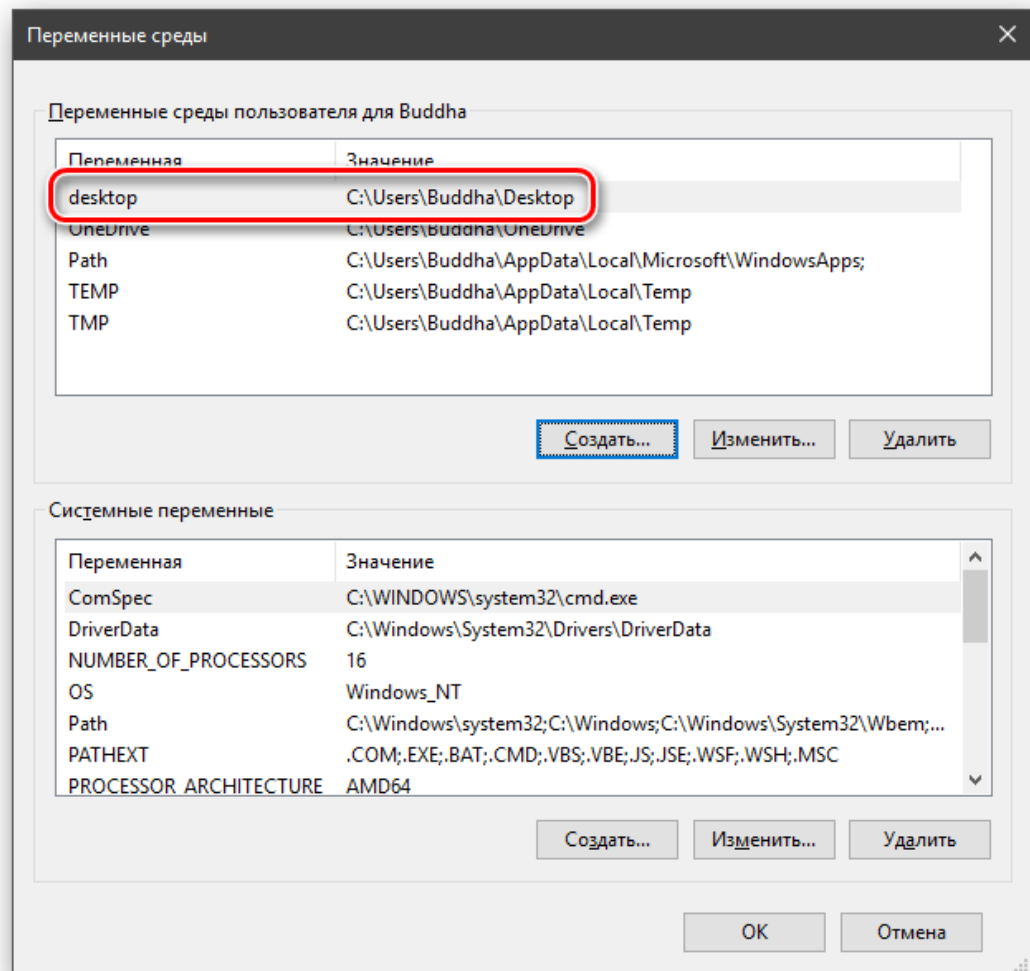


- Нажимаем **ОК**. Повторяем это действие во всех открытых окнах (см. выше).



5. Перезапускаем **Проводник** и консоль или целиком систему.

6. Готово, новая переменная создана, увидеть ее можно в соответствующем списке.



Для примера переделаем команду, которую мы использовали для получения списка (самая первая в статье). Теперь нам вместо

```
set > %homepath%\desktop\set.txt
```

потребуется ввести только

```
set > %desktop%\set.txt
```

Использование переменных окружения позволяет значительно сэкономить время при написании скриптов или взаимодействии с системной консолью. Еще одним плюсом является оптимизация создаваемого кода. Имейте в виду, что созданные вами переменные отсутствуют на других компьютерах, и сценарии (скрипты, приложения) с их использованием работать не будут, поэтому перед тем, как передавать файлы другому пользователю, необходимо уведомить его об этом и предложить создать соответствующий элемент в своей системе.

## Переменные окружения в 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.

На этом уроке мы научимся работать с переменными окружения и оболочки.

### Переменные окружения и переменные оболочки

Переменные можно разделить на две основные категории:

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

**Переменные оболочки** — это переменные, которые применяются только к текущему экземпляру оболочки. Каждая оболочка, например, [bash](#) или [zsh](#), имеет свой собственный набор внутренних переменных.

Все переменные имеют следующий формат:

```
KEY=значение1KEY="какое-то другое значение"KEY=значение1:значение2
```

При этом также следует придерживаться определенных правил:

Имена переменных чувствительны к регистру, поэтому переменные окружения должны иметь имена в верхнем регистре.

При присвоении переменной нескольких значений они должны быть разделены символом `:`.

Вокруг символа `=` не должно быть пробелов.

**Существует несколько команд, с помощью которых вы можете взаимодействовать с переменными окружения и оболочки:**

**команда `env`** — позволяет запускать другую программу в пользовательском окружении без изменения в текущем окружении. При использовании без аргумента выведет список переменных текущего окружения;

**команда `printenv`** — выводит список всех переменных окружения (или какую-то отдельно заданную переменную);

**команда `set`** — устанавливает переменные оболочки. При использовании без аргумента выведет список всех переменных, включая переменные окружения и переменные оболочки, а также функции оболочки;

**команда `unset`** — удаляет переменные оболочки и переменные окружения;

команда **export** — создает переменную окружения.

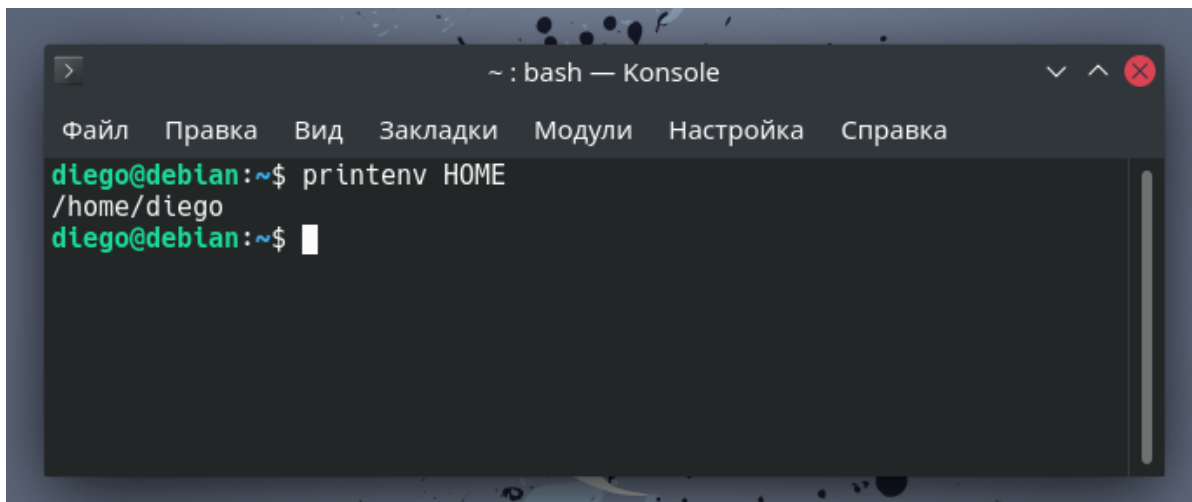
## Поиск и вывод переменных окружения

Наиболее часто используемая команда для вывода переменных окружения — `printenv`. Если команде в качестве аргумента передать имя переменной, то будет отображено значение только этой переменной. Если же вызвать `printenv` без аргументов, то выведется построчный список всех переменных окружения.

Например, чтобы отобразить значение переменной `HOME`, вы должны использовать команду:

```
$ printenv HOME
```

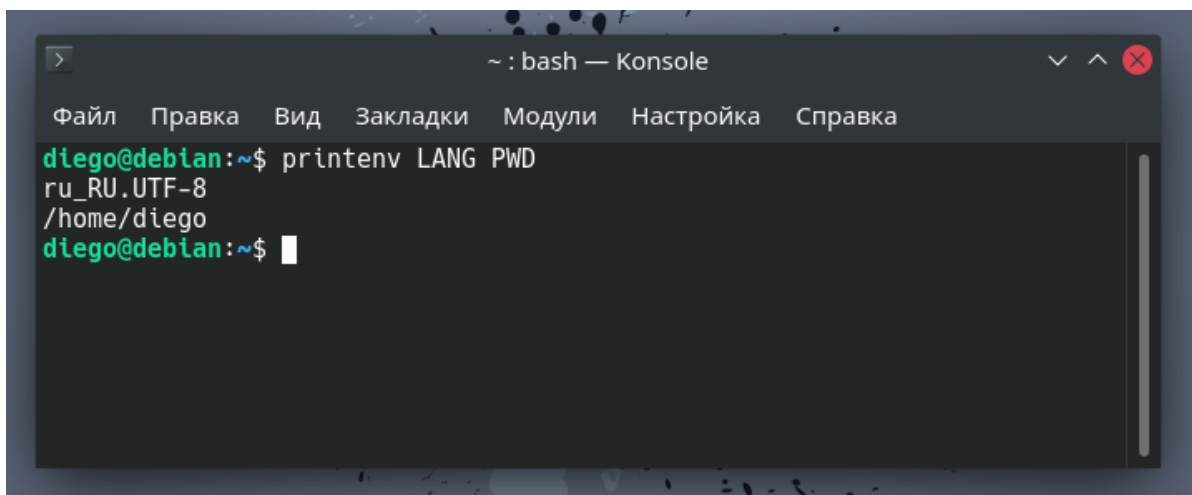
В результате вы увидите путь к домашнему каталогу текущего пользователя:



```
> ~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ printenv HOME
/home/diego
diego@debian:~$
```

Вы также можете передать команде `printenv` сразу несколько аргументов, например:

```
$ printenv LANG PWD
```



```
> ~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ printenv LANG PWD
ru_RU.UTF-8
/home/diego
diego@debian:~$
```

Если вы запустите команду `printenv` или `env` без каких-либо аргументов, то они покажут список всех переменных окружения:

```
$ printenv
```

```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/debian:@/tmp/.ICE-unix/988,unix/debian:/tmp/.ICE-unix/988
WINDOWID=4194311
QT_ACCESSIBILITY=1
COLORTERM=truecolor
XDG_SESSION_PATH=/org/freedesktop/DisplayManager/Session1
LANGUAGE=
D_DISABLE_RT_SCREEN_SCALE=1
SSH_AUTH_SOCK=/tmp/ssh-vNc0Xum0r057/agent.863
SHELL_SESSION_ID=8cebe2ca962f4edd940af8632b80666c
DESKTOP_SESSION=plasma
SSH_AGENT_PID=912
GTK_RC_FILES=/etc/gtk/gtkrc:/home/diego/.gtkrc:/home/diego/.config/gtkrc
XCURSOR_SIZE=24
XDG_SEAT=seat0
PWD=/home/diego
XDG_SESSION_DESKTOP=KDE
LOGNAME=diego
XDG_SESSION_TYPE=x11
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
XAUTHORITY=/home/diego/.Xauthority
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/diego/.gtkrc-2.0:/home/diego/.config/gtkrc-2.0
HOME=/home/diego
LANG=ru_RU.UTF-8
LS_COLORS=rs=0:di=01;34:ln=01;36:mh=00:pi=40;33:so=01;35:do=01;35:bd=40;33;01:cd=40;33;01:
or=40;31;01:mi=00:su=37;41:sg=30;43:ca=30;41:tw=30;42:ow=34;42:st=37;44:ex=01;32:*.tar=01;
```

Ниже приведены некоторые из наиболее распространенных **переменных окружения**:

- **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).

Наиболее распространенные **переменные оболочки**:

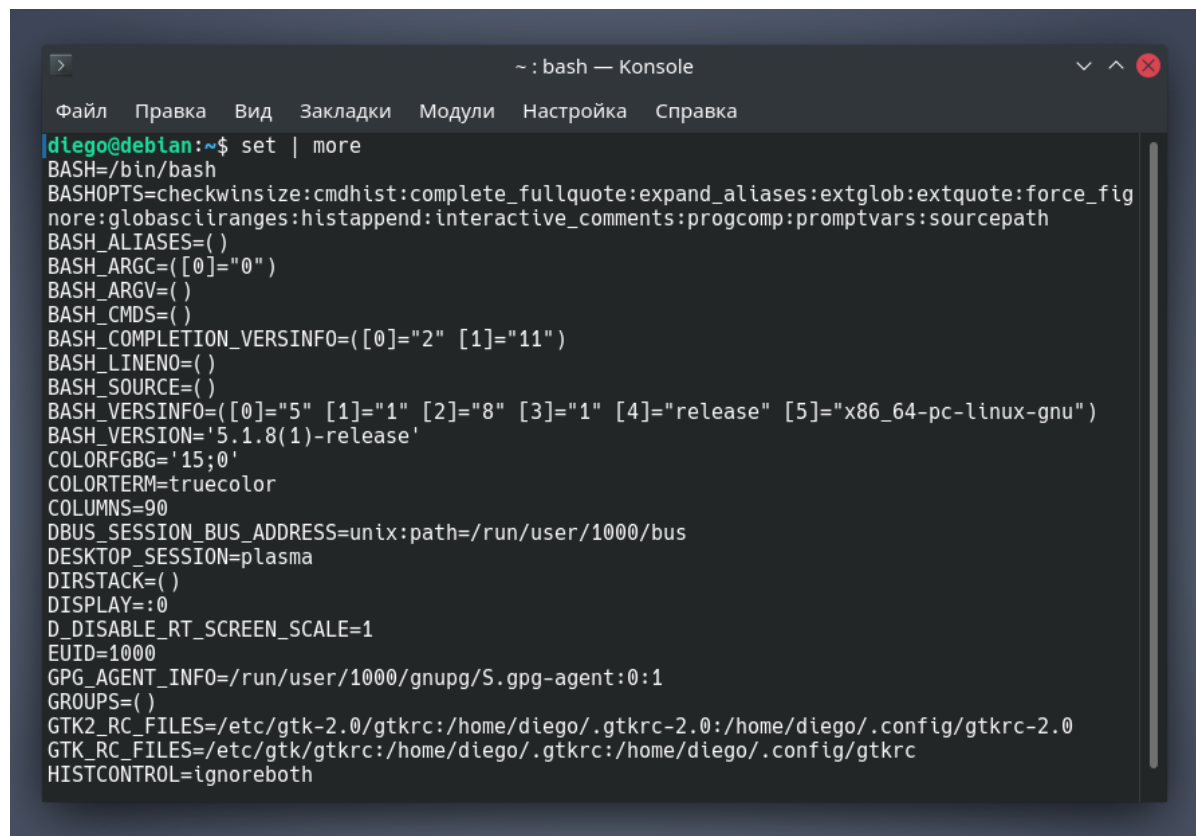
- **BASHOPTS** — список задействованных параметров оболочки, разделенных двоеточием.
- **BASH\_VERSION** — версия запущенной оболочки `bash`.
- **COLUMNS** — количество столбцов, которые используются для отображения выходных данных.
- **DIRSTACK** — стек директорий, к которому можно применять команды `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.
- `SHELLOPTS` — параметры оболочки, которые можно устанавливать с помощью команды `set`.
- `UID` — идентификатор текущего пользователя.

Команды `printenv` и `env` выводят только переменные окружения. Если вы хотите получить список всех переменных, включая переменные (и функции) оболочки, то можете использовать команду `set`:

```
$ set
```



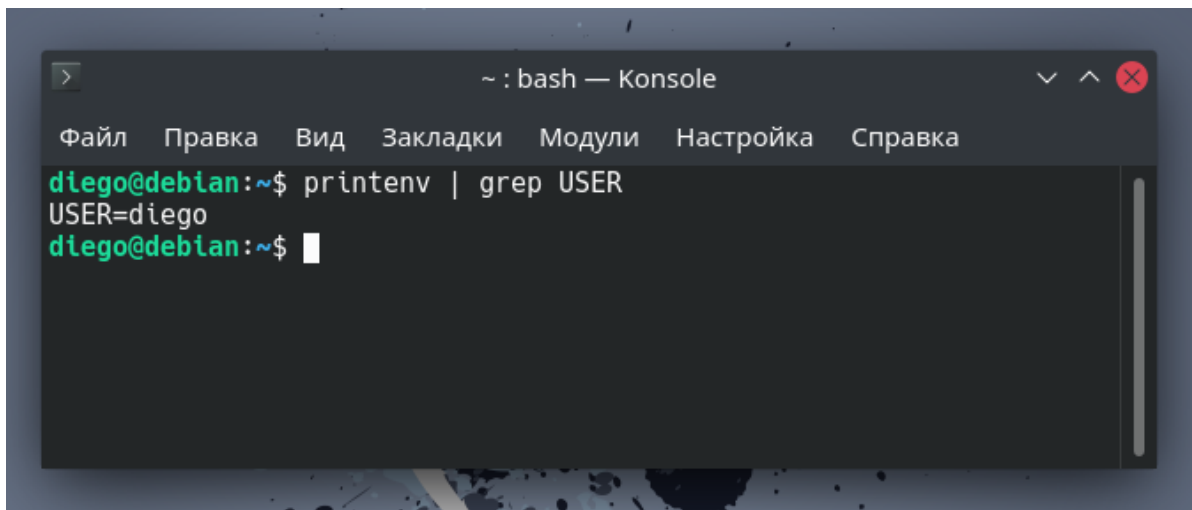
```
~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ set | more
BASH=/bin/bash
BASHOPTS=checkwinsize:cmdhist:complete_fullquote:expand_aliases:extglob:extquote:force_ig
nore:globasciiranges:histappend:interactive_comments:progcomp:promptvars:sourcepath
BASH_ALIASES=( )
BASH_ARGC=( [0]="0" )
BASH_ARGV=( )
BASH_CMDS=( )
BASH_COMPLETION_VERSION=( [0]="2" [1]="11" )
BASH_LINENO=( )
BASH_SOURCE=( )
BASH_VERSION=( [0]="5" [1]="1" [2]="8" [3]="1" [4]="release" [5]="x86_64-pc-linux-gnu" )
BASH_VERSION='5.1.8(1)-release'
COLORFGBG='15;0'
COLORTERM=truecolor
COLUMNS=90
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
DESKTOP_SESSION=plasma
DIRSTACK=( )
DISPLAY=:0
D_DISABLE_RT_SCREEN_SCALE=1
EUID=1000
GPG_AGENT_INFO=/run/user/1000/gnupg/S.gpg-agent:0:1
GROUPS=( )
GTK2_RC_FILES=/etc/gtk-2.0/gtkrc:/home/diego/.gtkrc-2.0:/home/diego/.config/gtkrc-2.0
GTK_RC_FILES=/etc/gtk/gtkrc:/home/diego/.gtkrc:/home/diego/.config/gtkrc
HISTCONTROL=ignoreboth
```

Команда отобразит список всех переменных. Он довольно большой, поэтому я заранее перенаправил вывод в команду `more`.

Чтобы найти все переменные, содержащие заданную строку, используйте команду `grep`:

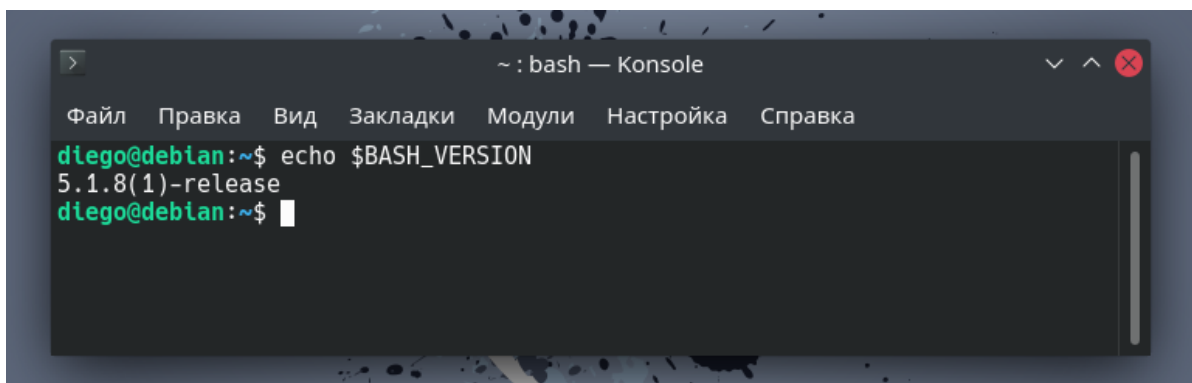
```
$ printenv | grep [ИМЯ_ПЕРЕМЕННОЙ]
```

Ниже представлен пример поиска переменных, в названии которых содержится строка `USER`:

A terminal window titled '~ : bash — Konsole' with a menu bar containing 'Файл', 'Правка', 'Вид', 'Закладки', 'Модули', 'Настройка', and 'Справка'. The prompt is 'diego@debian:~\$'. The command 'printenv | grep USER' is entered, and the output 'USER=diego' is displayed. The prompt returns to 'diego@debian:~\$'.

Для отображения переменных оболочки также можно использовать команду `echo`. Например, чтобы вывести в терминал значение переменной `BASH_VERSION`, вы должны выполнить:

```
$ echo $BASH_VERSION
```

A terminal window titled '~ : bash — Konsole' with a menu bar containing 'Файл', 'Правка', 'Вид', 'Закладки', 'Модули', 'Настройка', and 'Справка'. The prompt is 'diego@debian:~\$'. The command 'echo \$BASH\_VERSION' is entered, and the output '5.1.8(1)-release' is displayed. The prompt returns to 'diego@debian:~\$'.

## Установка переменных оболочки

Чтобы создать новую переменную оболочки с именем, например, `NEW_VAR` и значением `Ravesli.com`, просто введите:

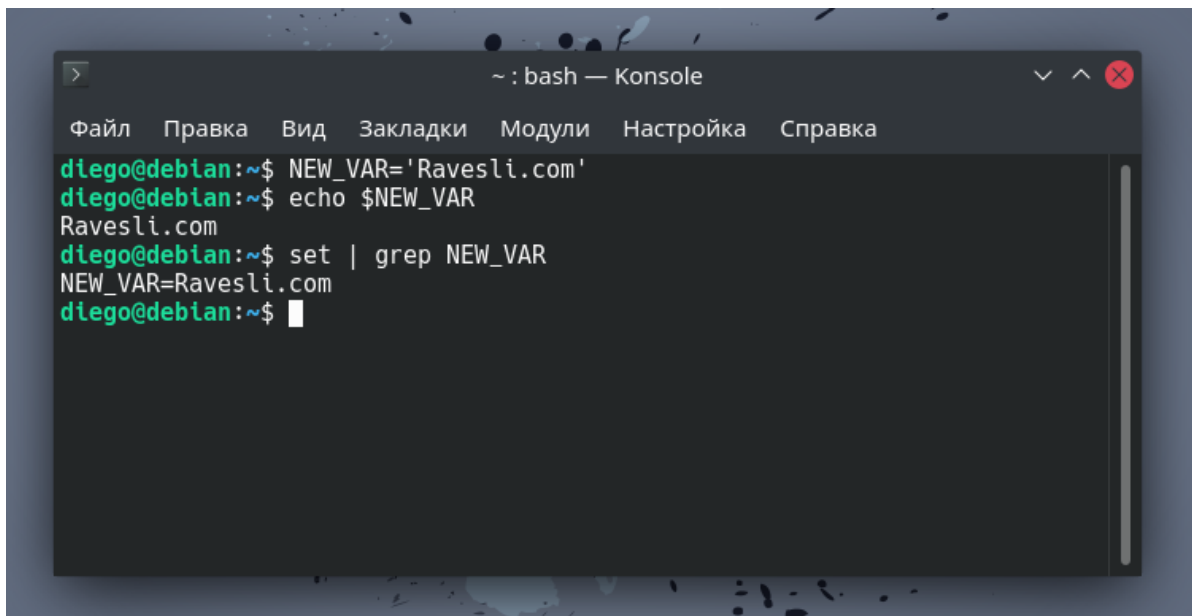
```
$ NEW_VAR='Ravesli.com'
```

Вы можете убедиться, что переменная действительно была создана, с помощью команды `echo`:

```
$ echo $NEW_VAR
```

либо

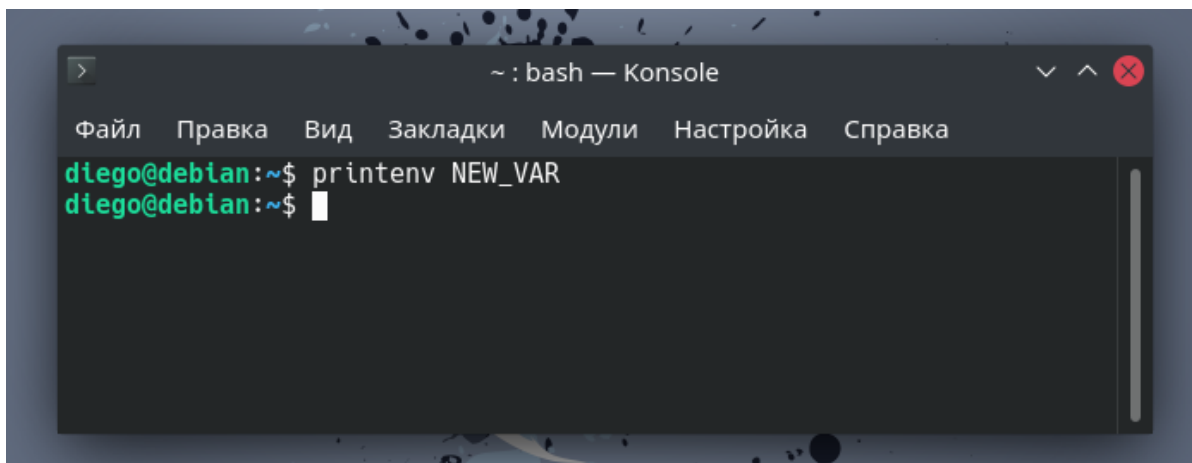
```
$ set | grep NEW_VAR
```

A terminal window titled '~ : bash — Konsole' with a menu bar (Файл, Правка, Вид, Закладки, Модули, Настройка, Справка). The prompt is 'diego@debian:~\$'. The user enters 'NEW\_VAR='Ravesli.com'', followed by 'echo \$NEW\_VAR' which outputs 'Ravesli.com'. Then 'set | grep NEW\_VAR' is entered, outputting 'NEW\_VAR=Ravesli.com'.

```
> ~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ NEW_VAR='Ravesli.com'
diego@debian:~$ echo $NEW_VAR
Ravesli.com
diego@debian:~$ set | grep NEW_VAR
NEW_VAR=Ravesli.com
diego@debian:~$
```

Используйте команду `printenv`, чтобы проверить, является ли наша переменная переменной окружения:

```
$ printenv NEW_VAR
```

A terminal window titled '~ : bash — Konsole' with a menu bar. The prompt is 'diego@debian:~\$'. The user enters 'printenv NEW\_VAR', and the output is empty.

```
> ~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ printenv NEW_VAR
diego@debian:~$
```

Вывод команды оказался пустым, что говорит нам о том, что созданная нами переменная не является переменной окружения.

Вы можете попробовать вывести значение переменной в новой оболочке, но вывод также будет пустым:

```
$ bash -c 'echo $NEW_VAR'
```

## Установка переменных окружения

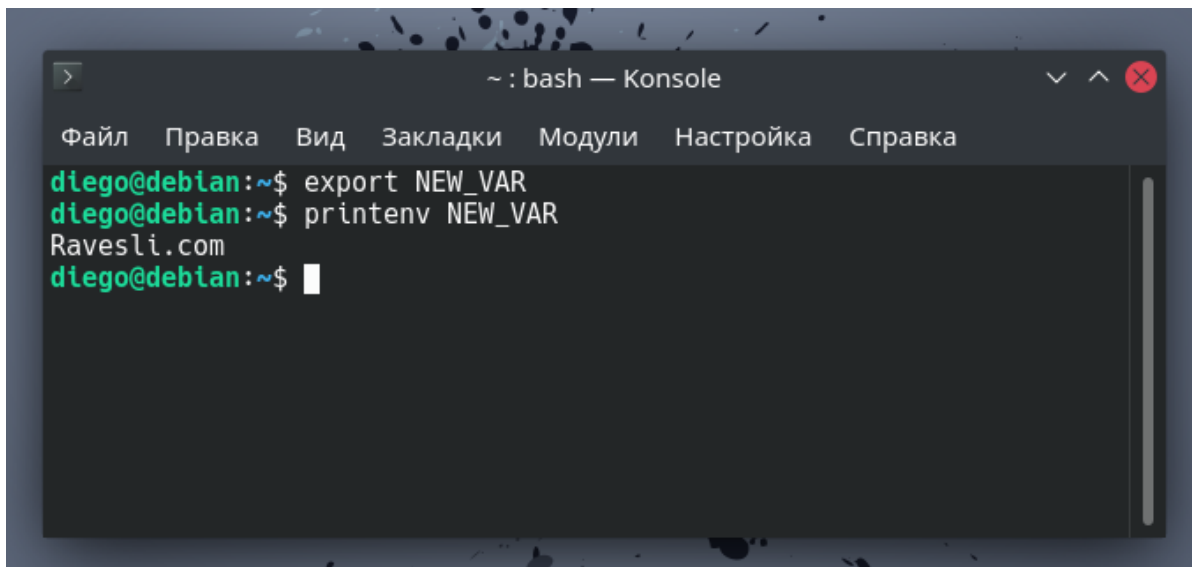
Команда **export** используется для задания переменных окружения. С помощью данной команды мы экспортируем указанную переменную, в результате чего она будет видна во всех вновь запускаемых дочерних командных оболочках. Переменные такого типа принято называть *внешними*.

Для создания переменной окружения экспортируем нашу недавно созданную переменную оболочки:

```
$ export NEW_VAR
```

Проверяем результат, действительно ли мы создали переменную окружения:

```
$ printenv NEW_VAR
```



На этот раз, если вы попытаетесь отобразить переменную в новой оболочке, получите её значение:

```
$ bash -c 'echo $NEW_VAR'
```

Результат:

```
Ravesli.com
```

Вы также можете использовать и следующую конструкцию для создания переменной окружения:

```
$ export MY_NEW_VAR="My New Var"
```

**Примечание:** Созданные подобным образом переменные окружения доступны только в текущем сеансе. Если вы откроете новую оболочку или выйдете из системы, то все переменные будут потеряны.

## Как сделать переменные окружения постоянными?

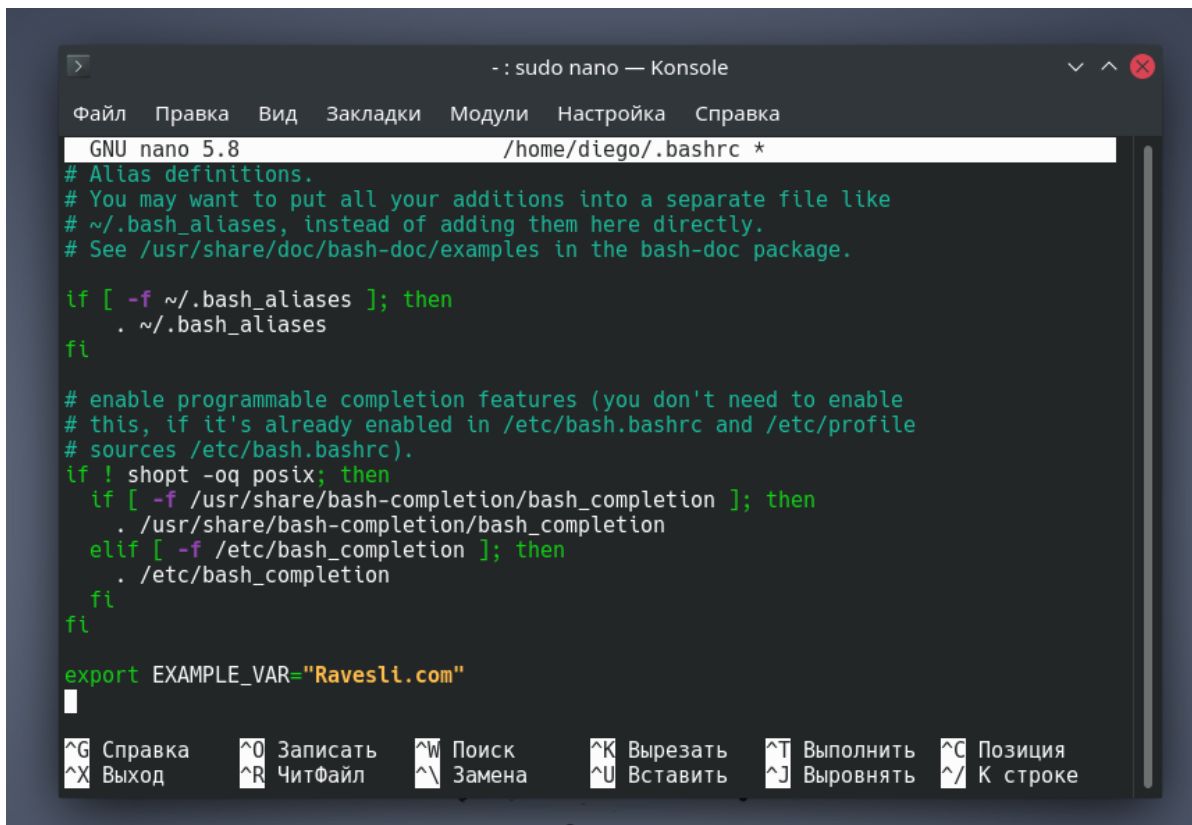
Если вы хотите, чтобы переменная сохранялась после закрытия сеанса оболочки, то необходимо прописать её в специальном файле. Прописать переменную можно как для текущего пользователя, так и для всех пользователей.

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

```
$ sudo nano ~/.bashrc
```

Для каждой переменной, которую вы хотите сделать постоянной, добавь в конец файла строку, используя следующий синтаксис:

```
export [ИМЯ_ПЕРЕМЕННОЙ]=[ЗНАЧЕНИЕ_ПЕРЕМЕННОЙ]
```



```
GNU nano 5.8 /home/diego/.bashrc *
# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

export EXAMPLE_VAR="Ravesli.com"
```

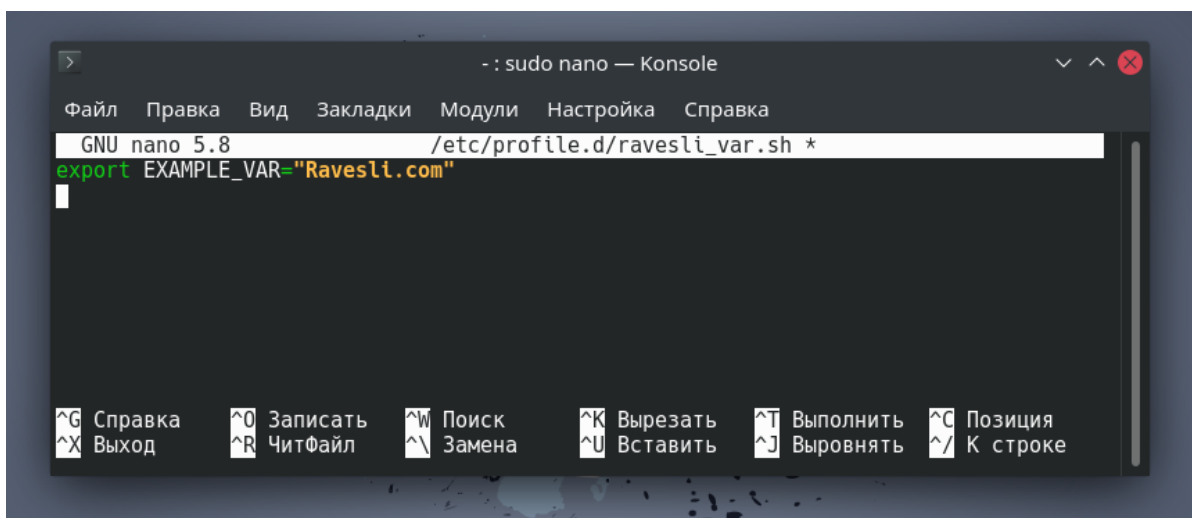
Сохраните и закройте файл. Изменения будут применены после перезапуска оболочки. Если вы хотите применить изменения во время текущего сеанса, то используйте команду `source` :

```
$ source ~/.bashrc
```

Чтобы задать постоянные переменные окружения для всех пользователей, создайте `.sh`-файл в каталоге `/etc/profile.d`.**\*\*d**:

```
$ sudo nano /etc/profile.d/[имя_файла].sh
```

Синтаксис добавления переменных в файл такой же, как и в случае с файлом `.bashrc`:



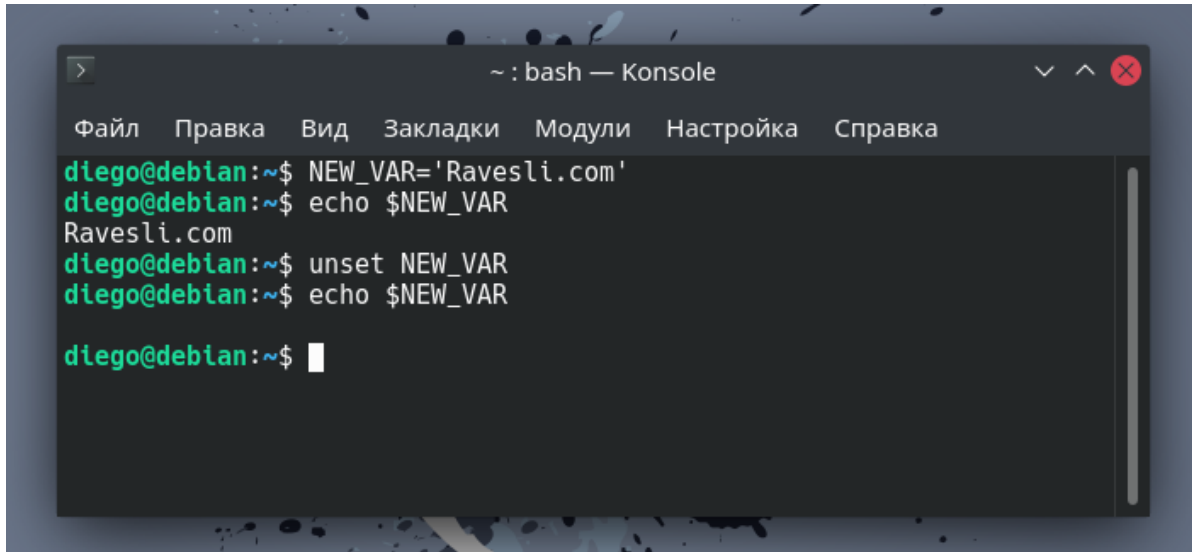
```
GNU nano 5.8 /etc/profile.d/ravesli_var.sh *
export EXAMPLE_VAR="Ravesli.com"
```

Сохраните и закройте файл. Изменения будут применены при следующем входе в систему.

## Удаление переменных

Чтобы полностью удалить переменную любого типа, используйте команду **unset**:

```
$ unset NEW_VAR
```



```
> ~ : bash — Konsole
Файл  Правка  Вид  Закладки  Модули  Настройка  Справка
diego@debian:~$ NEW_VAR='Ravesli.com'
diego@debian:~$ echo $NEW_VAR
Ravesli.com
diego@debian:~$ unset NEW_VAR
diego@debian:~$ echo $NEW_VAR

diego@debian:~$
```

Переменные окружения и оболочки всегда присутствуют в сеансах оболочки и могут быть очень полезны. Они позволяют родительским процессам устанавливать детали конфигурации для своих дочерних процессов и являются способом установки определенных параметров без использования отдельных файлов.

Это дает много преимуществ в конкретных ситуациях. Например, некоторые механизмы развертывания (деплоймента) полагаются на переменные окружения для настройки информации аутентификации. Переменные окружения и оболочки позволяют хранить эти данные не в файлах, которые могут быть просмотрены посторонними лицами.

Существует множество других распространенных сценариев, в которых вам нужно будет прочитать или изменить параметры/данные вашего окружения или оболочки. Теперь вы знаете как это можно сделать.

## Переменные среды и поведение Python

### `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` - это каталоги, зависящие от установки, оба каталога по умолчанию - `/usr/local`.

Когда для `PYTHONHOME` задан один каталог, его значение заменяет `prefix` и `exec_prefix`. Чтобы указать для них разные значения, установите для `PYTHONHOME` значение `prefix:exec_prefix`.

### `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.

Помимо обычных каталогов, отдельные записи `PYTHONPATH` могут относиться к zip-файлам, содержащим чистые модули Python в исходной или скомпилированной форме. Модули расширения нельзя импортировать из zip-файлов.

Путь поиска по умолчанию зависит от установки Python, но обычно начинается с префикса `/lib/pythonversion`. Он всегда добавляется к `PYTHONPATH`.

### **PYTHONSTARTUP:**

Если переменная среды `PYTHONSTARTUP` это имя файла, то команды Python в этом файле выполняются до отображения первого приглашения в интерактивном режиме. Файл выполняется в том же пространстве имен, в котором выполняются интерактивные команды, так что определенные или импортированные в нем объекты можно использовать без квалификации в интерактивном сеансе.

При запуске вызывает событие аудита `cpython.run_startup` с именем файла в качестве аргумента.

### **PYTHONOPTIMIZE:**

Если в переменной среды `PYTHONOPTIMIZE` задана непустая строка, это эквивалентно указанию параметра `-O`. Если установлено целое число, то это эквивалентно указанию `-OO`.

### **PYTHONBREAKPOINT:**

Если переменная среды `PYTHONBREAKPOINT` установлена, то она определяет вызываемый объект с помощью точечной нотации. Модуль, содержащий вызываемый объект, будет импортирован, а затем вызываемый объект будет запущен реализацией по умолчанию `sys.breakpointhook()`, которая сама вызывается встроенной функцией `breakpoint()`. Если `PYTHONBREAKPOINT` не задан или установлен в пустую строку, то это эквивалентно значению `pdb.set_trace`. Установка этого значения в строку `0` приводит к тому, что стандартная реализация `sys.breakpointhook()` ничего не делает, кроме немедленного возврата.

### **PYTHONDEBUG:**

Если значение переменной среды `PYTHONDEBUG` непустая строка, то это эквивалентно указанию опции `-d`. Если установлено целое число, то это эквивалентно многократному указанию `-dd`.

### **PYTHONINSPECT:**

Если значение переменной среды `PYTHONINSPECT` непустая строка, то это эквивалентно указанию параметра `-i`.

Эта переменная также может быть изменена кодом Python с помощью `os.environ` для принудительного режима проверки при завершении программы.

### **PYTHONUNBUFFERED:**

Если значение переменной среды `PYTHONUNBUFFERED` непустая строка, то это эквивалентно указанию параметра `-u`.

## PYTHONVERBOSE :

Если значение переменной среды `PYTHONVERBOSE` непустая строка, то это эквивалентно указанию опции `-v`. Если установлено целое число, это эквивалентно многократному указанию `-v`.

## PYTHONCASEOK :

Если значение переменной среды `PYTHONCASEOK` установлено, то Python игнорирует регистр символов в операторах импорта. Это работает только в Windows и OS X.

## PYTHONDONTWRITEBYTECODE :

Если значение переменной среды `PYTHONDONTWRITEBYTECODE` непустая строка, то Python не будет пытаться писать файлы `.рус` при импорте исходных модулей. Это эквивалентно указанию параметра `-B`.

## PYTHONPYCACHEPREFIX :

Если значение переменной среды `PYTHONPYCACHEPREFIX` установлено, то Python будет записывать файлы `.рус` в зеркальном дереве каталогов по этому пути, а не в каталогах `__pycache__` в исходном дереве. Это эквивалентно указанию параметра `-x pycache_prefix=PATH`.

## PYTHONHASHSEED :

Если значение переменной среды `PYTHONHASHSEED` не установлено или имеет значение `random`, то случайное значение используется для заполнения хэшей объектов `str` и `bytes`.

Если для `PYTHONHASHSEED` задано целочисленное значение, то оно используется как фиксированное начальное число для генерации `hash()` типов, охватываемых рандомизацией хэша.

Цель - разрешить повторяемое хеширование, например, для самотестирования самого интерпретатора, или позволить кластеру процессов Python совместно использовать хеш-значения.

Целое число должно быть десятичным числом в диапазоне `[0,4294967295]`. Указание значения `0` отключит рандомизацию хэша.

## PYTHONIOENCODING :

Если значение переменной среды `PYTHONIOENCODING` установлено до запуска интерпретатора, то оно переопределяет кодировку, используемую для `stdin / stdout / stderr`, в синтаксисе `encodingname:errorhandler`. И имя кодировки `encodingname`, и части `:errorhandler` являются необязательными и имеют то же значение, что и в функции `str.encode()`.

Для `stderr` часть `:errorhandler` игнорируется, а обработчик всегда будет заменять обратную косую черту.

## PYTHONNOUSERSITE :

Если значение переменной среды `PYTHONNOUSERSITE` установлено, то Python не будет добавлять пользовательский каталог `site-packages` в переменную `sys.path`.



## PYTHONUSERBASE:

Переменная среды `PYTHONUSERBASE` определяет базовый каталог пользователя, который используется для вычисления пути к каталогу пользовательских пакетов сайта `site-packages` и путей установки `Distutils` для `python setup.py install --user`.

## PYTHONWARNINGS:

Переменная среды `PYTHONWARNINGS` эквивалентна опции `-W`. Если она установлена в виде строки, разделенной запятыми, то это эквивалентно многократному указанию `-W`, при этом фильтры, расположенные позже в списке, имеют приоритет над фильтрами ранее в списке.

В простейших настройках определенное действие безоговорочно применяется ко всем предупреждениям, выдаваемым процессом (даже к тем, которые по умолчанию игнорируются):

- `PYTHONWARNINGS=default` - предупреждает один раз для каждого вызова;
- `PYTHONWARNINGS=error` - преобразовывает в исключения;
- `PYTHONWARNINGS=always` - предупреждает каждый раз;
- `PYTHONWARNINGS=module` - предупреждает один раз для каждого вызванного модуля;
- `PYTHONWARNINGS=once` - предупреждает один раз для каждого процесса Python;
- `PYTHONWARNINGS=ignore` - никогда не предупреждает.

## PYTHONFAULTHANDLER:

Если значение переменной среды `PYTHONFAULTHANDLER` непустая строка, то при запуске вызывается `faulthandler.enable()`: устанавливается обработчик сигналов `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` и `SIGILL`, чтобы вывести данные трассировки Python. Это эквивалентно опции обработчика ошибок `-X`.

## PYTHONTRACEMALLOC:

Если значение переменной среды `PYTHONTRACEMALLOC` непустая строка, то начнется отслеживание выделения памяти Python с помощью модуля `tracemalloc`. Значение переменной - это максимальное количество кадров, хранящихся в обратной трассировке `trace`. Например, `PYTHONTRACEMALLOC=1` сохраняет только самый последний кадр.

## PYTHONPROFILEIMPORTTIME:

Если значение переменной среды `PYTHONPROFILEIMPORTTIME` непустая строка, то Python покажет, сколько времени занимает каждый импорт. Это в точности эквивалентно установке `-X importtime` в командной строке.

## PYTHONASYNCIODEBUG:

Если значение переменной среды `PYTHONASYNCIODEBUG` непустая строка, то включается режим отладки модуля `asyncio`.

## PYTHONMALLOC:

Переменная `PYTHONMALLOC` задает распределители памяти Python и/или устанавливает отладочные хуки.

Задаёт семейство распределителей памяти, используемых Python:

- `default`: использует распределители памяти по умолчанию.

- `malloc`: использует функцию `malloc()` библиотеки `C` для всех доменов (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: использует распределитель `pymalloc` для доменов `PYMEM_DOMAIN_MEM` и `PYMEM_DOMAIN_OBJ` и использует функцию `malloc()` для домена `PYMEM_DOMAIN_RAW`.

Устанавливает хуки отладки:

- `debug`: устанавливает хуки отладки поверх распределителей памяти по умолчанию.
- `malloc_debug`: то же, что и `malloc`, но также устанавливает отладочные хуки.
- `pymalloc_debug`: то же, что и `pymalloc`, но также устанавливает отладочные хуки.

## PYTHONMALLOCSTATS:

Если значение переменной среды `PYTHONMALLOCSTATS` непустая строка, то Python будет печатать статистику распределителя памяти `pymalloc` каждый раз, когда создается новая область объекта `pymalloc`, а также при завершении работы.

Эта переменная игнорируется, если переменная среды `PYTHONMALLOC` используется для принудительного использования распределителя `malloc()` библиотеки `C` или если Python настроен без поддержки `pymalloc`.

## PYTHONLEGACYWINDOWSFSENCODING:

Если значение переменной среды Python `PYTHONLEGACYWINDOWSFSENCODING` непустая строка, то кодировка файловой системы по умолчанию и режим ошибок вернутся к своим значениям `mbcs` и `replace` до версии Python 3.6 соответственно. В противном случае используются новые значения по умолчанию `utf-8` и `surrogatepass`.

## PYTHONLEGACYWINDOWSSSTDIO:

Если значение переменной среды `PYTHONLEGACYWINDOWSSSTDIO` непустая строка, то новые средства чтения и записи консоли не используются. Это означает, что символы Unicode будут закодированы в соответствии с активной кодовой страницей консоли, а не с использованием `utf-8`.

Эта переменная игнорируется, если стандартные потоки перенаправляются в файлы или каналы, а не ссылаются на буферы консоли.

## PYTHONCOERCECLOCALE:

Если значение переменной среды `PYTHONCOERCECLOCALE` установлено в значение `0`, то это заставит основное приложение командной строки Python пропускать приведение устаревших локалей `C` и `POSIX` на основе `ASCII` к более функциональной альтернативе на основе `UTF-8`.

Если эта переменная не установлена или имеет значение, отличное от `0`, то переменная среды переопределения локали `LC_ALL` также не задана, а текущая локаль, указанная для категории `LC_CTYPE`, является либо локалью `C` по умолчанию, либо локалью `POSIX` явно основанной на `ASCII`, то Python `CLI` попытается настроить следующие локали для категории `LC_CTYPE` в порядке, указанном перед загрузкой среды выполнения интерпретатора:

- `C.UTF-8`,
- `C.utf8`,
- `UTF-8`.

Если установка одной из этих категорий локали прошла успешно, то переменная среды `LC_STYPE` также будет установлена соответствующим образом в текущей среде процесса до инициализации среды выполнения Python. Это гарантирует, что обновленный параметр будет виден как самому интерпретатору, так и другим компонентам, зависящим от локали, работающим в одном процессе (например, библиотеке GNU `readline`), и в subprocessах (независимо от того, работают ли эти процессы на интерпретаторе Python или нет), а также в операциях, которые запрашивают среду, а не текущую локаль `C` (например, собственный `locale.getDefaultLocale()` Python).

Настройка одного из этих языковых стандартов явно или с помощью указанного выше неявного принуждения языкового стандарта автоматически включает обработчик ошибок `surrogateescape` для `sys.stdin` и `sys.stdout` (`sys.stderr` продолжает использовать обратную косую черту, как и в любой другой локали). Это поведение обработки потока можно переопределить, используя `PYTHONIOENCODING`, как обычно.

Для целей отладки, установка `PYTHONCOERCECLOCALE=warn` приведет к тому, что Python будет выдавать предупреждающие сообщения на `stderr`, если активируется принуждение языкового стандарта или если языковой стандарт, который мог бы вызвать приведение, все еще активен при инициализации среды выполнения Python.

Также обратите внимание, что даже когда принуждение языкового стандарта отключено или когда не удастся найти подходящую целевую локаль, переменная среды `PYTHONUTF8` все равно будет активироваться по умолчанию в устаревших локалях на основе `ASCII`. Чтобы для системных интерфейсов интерпретатор использовал `ASCII` вместо `UTF-8`, необходимо обе переменные отключить.

### **`PYTHONDEVMODE` :**

Если значение переменной среды `PYTHONDEVMODE` непустая строка, то включится режим разработки Python, введя дополнительные проверки времени выполнения, которые слишком "дороги" для включения по умолчанию.

### **`PYTHONUTF8` :**

Если переменная среды `PYTHONUTF8` установлена в значение 1, то это включает режим интерпретатора `UTF-8`, где `UTF-8` используется как кодировка текста для системных интерфейсов, независимо от текущей настройки локали.

### **`PYTHONWARNDEFAULTENCODING` :**

Если для этой переменной среды задана непустая строка, то код будет выдавать `EncodingWarning`, когда используется кодировка по умолчанию, зависящая от локали.

### **`PYTHONTHREADDEBUG` :**

Если значение переменной среды `PYTHONTHREADDEBUG` установлено, то Python распечатает отладочную информацию о потоках.

Нужен Python, настроенный с параметром сборки `--with-pydebug`.

## PYTHONDUMPREFS:

Если значение переменной среды `PYTHONDUMPREFS` установлено, то Python будет сбрасывать объекты и счетчики ссылок, все еще живые после завершения работы интерпретатора.

## Чтение и запись переменных окружения в Python

Переменные окружения используются для изменения конфигурации системы. Результат работы многих приложений на Python зависит от значений определённых переменных окружения. Когда эти переменные изменяются, для получения прежнего результата скрипт Python требует корректировок, а это нежелательно. Эту проблему можно решить, считывая и изменяя значения нужных нам переменных в самом скрипте.

Это избавит нас от необходимости исправлять переменные среды вручную и сделает код безопаснее: будут спрятаны конфиденциальные данные, которые требуется присвоить переменной окружения (например, токен API).

В этом уроке мы рассмотрим способы установки и получения таких переменных средствами языка Python.

### Чтение переменных окружения на Python

Для начала потребуется импортировать модуль `os`, чтобы считывать переменные. Для доступа к переменным среды в Python используется объект `os.environ`. С его помощью программист может получить и изменить значения всех переменных среды. Далее мы рассмотрим различные способы чтения, проверки и присвоения значения переменной среды.

### Считываем одну или все переменные окружения

Следующий код позволяет прочитать и вывести все переменные окружения, а также определенную переменную. Для вывода имен и значений всех переменных используется цикл `for`. Затем выводится значение переменной `HOME`.

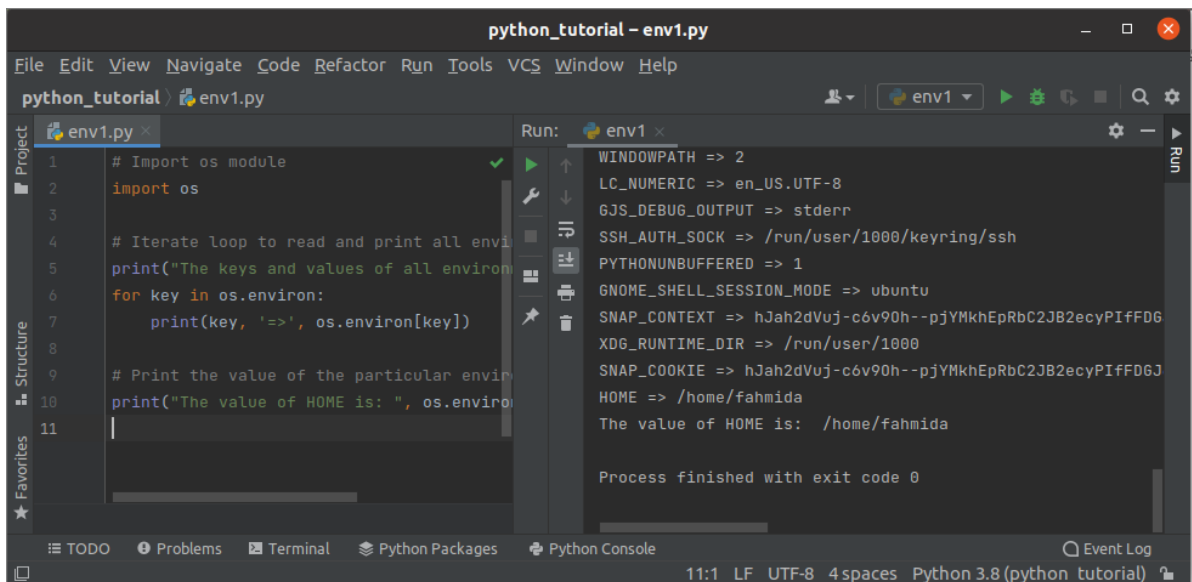
```
# Импортируем модуль os
import os

# Создаём цикл, чтобы вывести все переменные среды
print("The keys and values of all environment variables:")
for key in os.environ:
    print(key, '=>', os.environ[key])

# Выводим значение одной переменной
print("The value of HOME is: ", os.environ['HOME'])
```

После выполнения скрипта мы увидим следующий результат. Сперва был выведен список всех переменных окружения, а затем – значение переменной

`HOME`.



## Проверяем, присвоено ли значение переменной окружения

Давайте создадим Python-файл со следующим скриптом для проверки переменных. Для чтения значений переменных мы используем модуль `os`, а модуль `sys` — для прекращения работы приложения.

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.

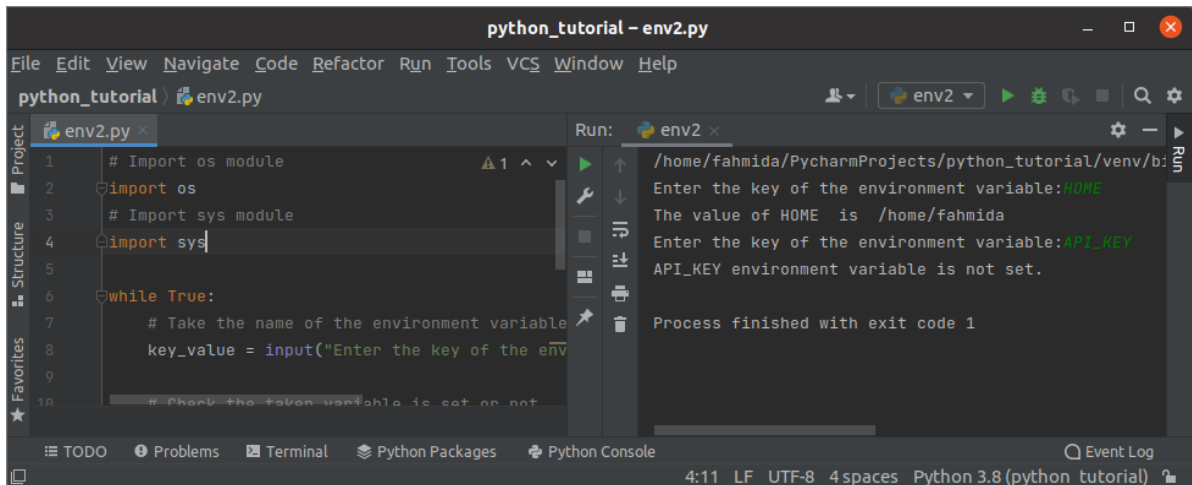
Если пользователь вводит имя переменной окружения, которой присвоено значение, это значение выводится, если же нет — выводится соответствующее сообщение и процесс останавливается.

```
# Импортируем модуль os
import os
# Импортируем модуль sys
import sys

while True:
    # Принимаем имя переменной среды
    key_value = input("Enter the key of the environment variable:")

    # Проверяем, инициализирована ли переменная
    try:
        if os.environ[key_value]:
            print(
                "The value of",
                key_value,
                " is ",
                os.environ[key_value]
            )
        # Если переменной не присвоено значение, то ошибка
    except KeyError:
        print(key_value, 'environment variable is not set.')
        # Завершаем процесс выполнения скрипта
        sys.exit(1)
```

На скрине вы видите результат работы скрипта. Первый раз было введено имя переменной, имеющей значение, а во второй раз — имя переменной, для которой значение не установлено. Согласно выводу, переменная `HOME` была инициализирована, и её значение вывелось в консоли. Переменной `API_KEY` не было задано значение, потому скрипт после вывода сообщения завершил работу.



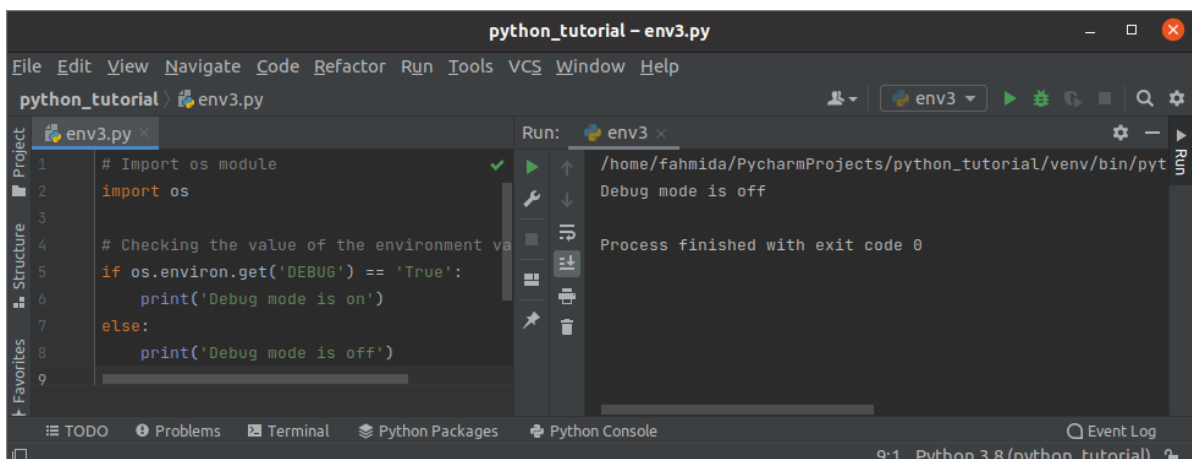
## Проверяем переменную на истинность

Создаём Python-файл со следующим кодом. Для проверки переменной `DEBUG` на истинность здесь используется функция `get()`. Программа выводит разные сообщения в зависимости от значения переменной.

```
# Импортируем модуль os
import os

# Проверяем значение переменной среды
if os.environ.get('DEBUG') == 'True':
    print('Debug mode is on')
else:
    print('Debug mode is off')
```

На скрине показан результат работы кода, если значение переменной `DEBUG` – `False`. Значение переменной можно изменить с помощью функции `setdefault()`, которую мы разберём в следующем разделе.



## Присваиваем значение переменной окружения

Для присвоения значения любой переменной среды используется функция `setDefault()`.

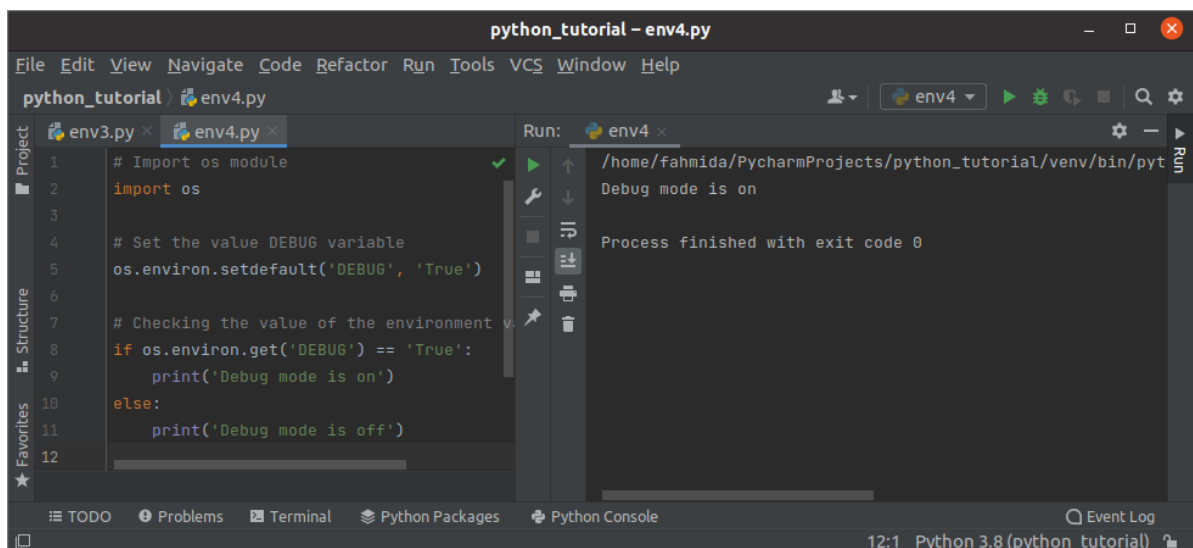
Давайте напишем код, чтобы с помощью функции `setDefault()` изменить значение переменной `DEBUG` на `True` (по умолчанию установлено `False`). После установки значения мы проверим его функцией `get()`.

Если мы сделали всё правильно, выведется сообщение «Режим отладки включен», в противном случае – «Режим отладки выключен».

```
# Импортируем модуль os
import os

# Задаём значение переменной DEBUG
os.environ.setdefault('DEBUG', 'True')
# Проверяем значение переменной
if os.environ.get('DEBUG') == 'True':
    print('Debug mode is on')
else:
    print('Debug mode is off')
```

Результат представлен ниже. Переменной `DEBUG` было присвоено значение `True`, и, соответственно, будет выведено сообщение «Режим отладки включен».



Значения переменных окружения можно считывать и изменять при помощи объекта `environ[]` модуля `os` либо путем использования функций `setDefault()` и `get()`.

В качестве ключа, по которому можно обратиться и получить либо присвоить значение переменной, в `environ[]` используется имя переменной окружения.

Функция `get()` используется для получения значения определённой переменной, а `setDefault()` – для инициализации.

**Пример 1.** Для примера 1 лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответствующую переменную окружения.

Для хранения имени файла данных будем использовать переменную окружения `WORKERS_DATA`. При этом сохраним возможность передавать имя файла данных через именованной параметр `--data`. Иными словами, если при запуске программы в командной строке не задан параметр `--data`, то имя файла данных должно быть взято из переменной окружения `WORKERS_DATA`.

Напишем программу для решения поставленной задачи.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import argparse
import json
import os
import sys
from datetime import date

def add_worker(staff, name, post, year):
    """
    Добавить данные о работнике.
    """
    staff.append(
        {
            "name": name,
            "post": post,
            "year": year
        }
    )

    return staff

def display_workers(staff):
    """
    Отобразить список работников.
    """
    # Проверить, что список работников не пуст.
    if staff:
        # Заголовок таблицы.
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        print(line)
        print(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",
                "Ф.И.О.",
                "Должность",
                "Год"
            )
        )
        print(line)

        # Вывести данные о всех сотрудниках.
```



```

        for idx, worker in enumerate(staff, 1):
            print(
                '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
                    idx,
                    worker.get('name', ''),
                    worker.get('post', ''),
                    worker.get('year', 0)
                )
            )
            print(line)

    else:
        print("Список работников пуст.")

def select_workers(staff, period):
    """
    Выбрать работников с заданным стажем.
    """
    # Получить текущую дату.
    today = date.today()

    # Сформировать список работников.
    result = []
    for employee in staff:
        if today.year - employee.get('year', today.year) >= period:
            result.append(employee)

    # Возвратить список выбранных работников.
    return result

def save_workers(file_name, staff):
    """
    Сохранить всех работников в файл JSON.
    """
    # Открыть файл с заданным именем для записи.
    with open(file_name, "w", encoding="utf-8") as fout:
        # Выполнить сериализацию данных в формат JSON.
        # Для поддержки кириллицы установим ensure_ascii=False
        json.dump(staff, fout, ensure_ascii=False, indent=4)

def load_workers(file_name):
    """
    Загрузить всех работников из файла JSON.
    """
    # Открыть файл с заданным именем для чтения.
    with open(file_name, "r", encoding="utf-8") as fin:
        return json.load(fin)

def main(command_line=None):
    # Создать родительский парсер для определения имени файла.
    file_parser = argparse.ArgumentParser(add_help=False)
    file_parser.add_argument(
        "-d",
        "--data",

```

```

        action="store",
        required=False,
        help="The data file name"
    )

    # Создать основной парсер командной строки.
    parser = argparse.ArgumentParser("workers")
    parser.add_argument(
        "--version",
        action="version",
        version="%prog)s 0.1.0"
    )

    subparsers = parser.add_subparsers(dest="command")

    # Создать субпарсер для добавления работника.
    add = subparsers.add_parser(
        "add",
        parents=[file_parser],
        help="Add a new worker"
    )
    add.add_argument(
        "-n",
        "--name",
        action="store",
        required=True,
        help="The worker's name"
    )
    add.add_argument(
        "-p",
        "--post",
        action="store",
        help="The worker's post"
    )
    add.add_argument(
        "-y",
        "--year",
        action="store",
        type=int,
        required=True,
        help="The year of hiring"
    )

    # Создать субпарсер для отображения всех работников.
    _ = subparsers.add_parser(
        "display",
        parents=[file_parser],
        help="Display all workers"
    )

    # Создать субпарсер для выбора работников.
    select = subparsers.add_parser(
        "select",
        parents=[file_parser],
        help="Select the workers"
    )
    select.add_argument(
        "-p",

```

```

        "--period",
        action="store",
        type=int,
        required=True,
        help="The required period"
    )

    # Выполнить разбор аргументов командной строки.
    args = parser.parse_args(command_line)

    # Получить имя файла.
    data_file = args.data
    if not data_file:
        data_file = os.environ.get("WORKERS_DATA")
    if not data_file:
        print("The data file name is absent", file=sys.stderr)
        sys.exit(1)

    # Загрузить всех работников из файла, если файл существует.
    is_dirty = False
    if os.path.exists(data_file):
        workers = load_workers(data_file)
    else:
        workers = []

    # Добавить работника.
    if args.command == "add":
        workers = add_worker(
            workers,
            args.name,
            args.post,
            args.year
        )
        is_dirty = True

    # Отобразить всех работников.
    elif args.command == "display":
        display_workers(workers)

    # Выбрать требуемых работников.
    elif args.command == "select":
        selected = select_workers(workers, args.period)
        display_workers(selected)

    # Сохранить данные в файл, если список работников был изменен.
    if is_dirty:
        save_workers(data_file, workers)

if __name__ == "__main__":
    main()

```

После установки значения переменной окружения `WORKERS_DATA` допустим запуск программы без указания имени файла. Например:

```
$ python workers.py add --name="Сидоров Сидор" -- post="Главный инженер" --  
year=2012
```

## Аппаратура и материалы

---

1. Компьютерный класс общего назначения с конфигурацией ПК не хуже рекомендованной для ОС Windows 10 с подключением к глобальной сети Интернет.
2. Операционная система Windows 10.
3. Система контроля версий Git.
4. Браузер для доступа к web-сервису GitHub, рекомендован к использованию Google Chrome.
5. Дистрибутив языка программирования Python, включающий набор популярных библиотек Anaconda.
6. Интегрированная среда разработки PyCharm Community Edition.

## Указания по технике безопасности

---

При работе на ЭВМ без разрешения руководителя занятия запрещается:

- подавать (снимать) напряжение на ПЭВМ и электрические розетки с распределительного щита;
- включать и выключать блоки питания ПЭВМ и мониторы;
- извлекать ПЭВМ из защитного кожуха;
- устранять неисправности, возникшие в ходе выполнения лабораторной работы.

## Методика и порядок выполнения работы

---

1. Изучить теоретический материал работы.
2. Создать общедоступный репозиторий на GitHub, в котором будет использована лицензия MIT и язык программирования Python.
3. Выполните клонирование созданного репозитория.
4. Дополните файл `.gitignore` необходимыми правилами для работы с IDE PyCharm.
5. Организуйте свой репозиторий в соответствии с моделью ветвления git-flow.
6. Создайте проект PyCharm в папке репозитория.
7. Проработайте примеры лабораторной работы. Создайте для них отдельные модули языка Python. Зафиксируйте изменения в репозитории.
8. Приведите в отчете скриншоты результатов выполнения примера при различных исходных данных вводимых с клавиатуры.
9. Зафиксируйте сделанные изменения в репозитории.
10. Приведите в отчете скриншоты работы программ решения индивидуальных заданий.
11. Зафиксируйте сделанные изменения в репозитории.
12. Добавьте отчет по лабораторной работе в *формате PDF* в папку *doc* репозитория. Зафиксируйте изменения.
13. Выполните слияние ветки для разработки с веткой *master/main*.
14. Отправьте сделанные изменения на сервер GitHub.
15. Отправьте адрес репозитория GitHub на электронный адрес преподавателя.

## Индивидуальные задания

---

## Задание 1

Для своего варианта лабораторной работы 2.17 добавьте возможность получения имени файла данных, используя соответствующую переменную окружения.

## Задание 2

Самостоятельно изучите работу с пакетом `python-dotenv`. Модифицируйте программу задания 1 таким образом, чтобы значения необходимых переменных окружения считывались из файла `.env`.

## Содержание отчета и его форма

---

Отчет по лабораторной работе оформляется электронно в формате PDF, должен содержать ответы на контрольные вопросы, ссылку на репозиторий с которым выполнялась работа, скриншоты IDE PyCharm, скриншоты результатов работы программ.

## Вопросы для защиты работы

---

1. Каково назначение переменных окружения?
2. Какая информация может храниться в переменных окружения?
3. Как получить доступ к переменным окружения в ОС Windows?
4. Каково назначение переменных `PATH` и `PATHEXT`?
5. Как создать или изменить переменную окружения в Windows?
6. Что представляют собой переменные окружения в ОС Linux?
7. В чем отличие переменных окружения от переменных оболочки?
8. Как вывести значение переменной окружения в Linux?
9. Какие переменные окружения Linux Вам известны?
10. Какие переменные оболочки Linux Вам известны?
11. Как установить переменные оболочки в Linux?
12. Как установить переменные окружения в Linux?
13. Для чего необходимо делать переменные окружения Linux постоянными?
14. Для чего используется переменная окружения `PYTHONHOME`?
15. Для чего используется переменная окружения `PYTHONPATH`?
16. Какие еще переменные окружения используются для управления работой интерпретатора Python?
17. Как осуществляется чтение переменных окружения в программах на языке программирования Python?
18. Как проверить, установлено или нет значение переменной окружения в программах на языке программирования Python?
19. Как присвоить значение переменной окружения в программах на языке программирования Python?