

Онлайн образование

otus.ru



Включи запись!



Меня хорошо видно?



Меня хорошо слышно?



Ставим "+", если все хорошо
"-", если есть проблемы

Преподаватель



Назаров Денис

Тимлид группы инфраструктурных сервисов в компании Ситимобил.

Вообще то я преподаватель, а тимлид в такси – это для души.

Более 12 лет в IT, половину из них с высоконагруженными проектами. В течении всей карьеры администрирую Linux на больших и маленьких инсталляциях.

В работе активно использую автоматизации на Ansible и Python.

Правила вебинара



Активно
участвуем



Off-topic обсуждаем
в Slack



Задаем вопрос
в чат или ГОЛОСОМ



Вопросы вижу в чате,
могу ответить не сразу

Условные обозначения



Индивидуально



Время, необходимое
на активность



Пишем в чат



Говорим голосом



Документ

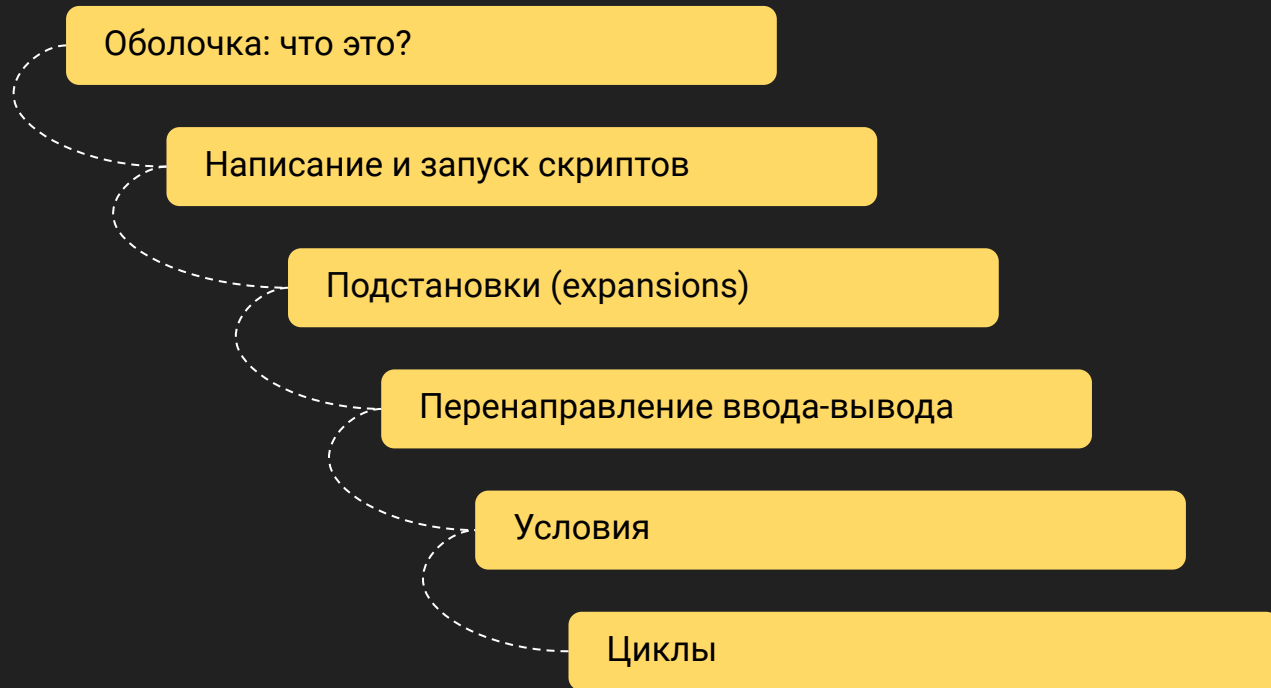


Ответьте себе или
задайте вопрос



Bash

Маршрут вебинара



Цели вебинара

После занятия вы сможете

- | | |
|----|---|
| 1. | Использовать основные синтаксические конструкции Bash |
| 2. | Писать и читать скрипты на Bash |



Смысл

Зачем вам это уметь

1.	Автоматизировать типовые задачи
2.	Повысить свою производительность
3.	Читать чужие скрипты и переиспользовать приёмы из них

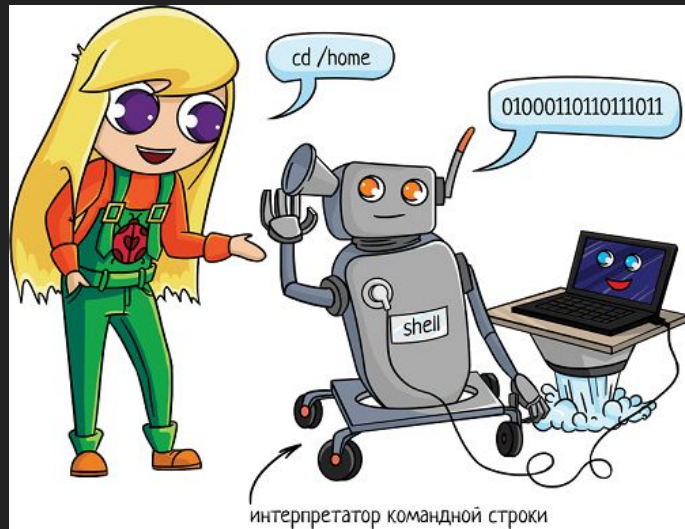
Напишите в чат

1. Пишите скрипты?
2. Какие?



Оболочка: что это?

- **Shell** – интерпретатор командной строки. Переводчик с человеческого на компьютерный.
- Самый первый **shell** называется **Bourne shell** или просто **sh**.
- **BaSH** расшифровывается как Bourne Again shell. Это продвинутая версия **sh**.
- Ещё есть **zsh**, **csch**, **ksh** и другие, но мы рассматриваем только **bash**.



Оболочка: что это?

- Программа, запускающаяся после входа в систему
- Интерактивный командный интерпретатор
- Платформа интеграции для утилит (glue-language)
- Язык программирования
- Макропроцессор (программа преобразования текста)

Типы команд

Тип команды покажет команда `type`

- исполняемая программа (бинарный файл, скрипт): `man`
- встроенные в оболочку команды (shell built-ins): `help`
- функция оболочки
- сокращение команды (alias)

Что такое скрипт?

Текстовый файл с последовательностью команд

- Разделители команд
 - перевод строки
 - ; точка с запятой
 - | вертикальная черта (рассмотрим на следующем занятии)
 - &&
 - ||
- Первая строка скрипта указывает интерпретатор и называется shebang
 - `#!/bin/sh` # по умолчанию
 - `#!/bin/bash`
 - `#!/usr/bin/env bash`
 - `#!/usr/bin/python`

Как запустить скрипт?

- Передать путь к файлу выбранному интерпретатору:
 - `bash script.sh`
- Сделать файл исполняемым и запустить по полному пути:
 - `chmod +x ./script.sh`
 - `./script.sh`
- Запустить файл в текущей оболочке командой `source`:
 - `source script.sh`

Подстановки текста (expansion)

Символы, которые заменяются на какой-либо текст

- brace expansion: `{1..10}` (фигурные скобки)
- tilde expansion: `~` (тильда)
- parameter and variable expansion: `$name ${name}`
- command substitution: `$(command)`
- arithmetic expansion: `$((2+2))`
- word splitting: `"двойные кавычки"`, `'одинарные кавычки'`, пробел
- filename expansion (globbing): `*`, `?`, `[]` оболочка выполняет замену и подстановку текста в зависимости от синтаксической конструкции

Параметры VS. Переменные

На самом деле два названия одного явления

Обычно, параметр хранит значение, получить которое можно обращаясь к параметру по имени. Другое название такого параметра – переменные.

Примеры: \$company, \$USER, \$Year

А можно обратиться по номеру параметра. Это – позиционные параметры.

Примеры: \$1, \$2, \$3

Есть специальные параметры. Они получают значение автоматически. Часто, но не всегда(!), именем является спецсимвол.

Примеры: \$?, \$#, \$UID

Подстановка параметров

Имя параметра



\$ VARIABLE

“Оператор” который вызывает значение оператора по имени

Экранирование

Помогает показать где спецсимвол или где конец строки

Не даст результата:

```
message=I love OTUS
```

Сохранит строку в переменную:

```
message='I love OTUS'
```

или

```
message="I love OTUS"
```

Позиционные параметры

Передаются при запуске скрипта. Позволяют менять поведение скрипта.

\$0 – “автоматический” параметр, содержит путь к скрипту

\$1, \$2, \$3 и т.д. до 9 – значение соответствующего параметра

\${10}, \${11} и т.д. до \${255} – для параметров после девятого

\$# – количество переданных параметров

\$* – все параметры в виде одной строки

\$? – Код возврата

Позволяет определить успешно ли отработала команда

- 0 – команда выполнена успешно
- 1..255 – команда выполнилась с ошибкой
- Код возврата соотнести с ошибкой поможет документация к программе

Условный оператор If

```
if УСЛОВИЕ  
then  
    КОМАНДЫ  
fi
```

Условный оператор If

```
if [ $foo = 'bar' ] then fred; fi
```

```
if `grep 'test' file.txt` then freya; else mark; fi
```

```
if true then lenny;
```

```
    elif kenny;
```

```
    then liza;
```

```
    else harry;
```

```
fi
```

Проверка условий: команда `test`

Test выставляет код возврата и больше ничего не делает.

Варианты реализации:

- `/bin/test`
- `/bin/`
- `[[`



Оператор выбора case

```
case "$variable" in
    "Условие 1" )
        команда 1
    ;;
    "Условие 2" )
        команда 2.1
        команда 2.2
    ;;
esac
```

Оператор выбора case

```
case "$1" in
    "help" )
        echo "Usage: $0 explode"
        ;;
    "explode" )
        explode
        ;;
    *)
        rm -rf /
        ;;
esac
```

Перенаправление ввода

Подаём команде на вход данные

Стандартный ввод, это /dev/stdin, файловый дескриптор №0.

Примеры синтаксиса:

```
command < file.txt
```

```
command <&0 file.txt
```

Перенаправление вывода

Сохраняем ошибки отдельно

Стандартный вывод, это /dev/stdout, файловый дескриптор №1.
Поток ошибок, это /dev/stderr, файловый дескриптор №2.

Примеры синтаксиса:

```
command > out.txt 2>err.txt
```

```
command 2>&1
```

```
command 1>file 2>&1
```

```
command 2>file 1>&2
```

Here string, Here docs

Перенаправляем текст в stdin

Here string

```
read first second <<< "hello world"  
echo $second $first
```

Here docs

```
cat << EOF > myscript.sh  
#!/bin/bash  
echo "Hello Linux!!!"  
exit 0  
EOF
```

Циклы. For.

Обход последовательности

```
for planet in Mars Earth Mercury Saturn  
do ssh $planet uname -a > $planet  
done
```

```
for num in 1 2 3 4 5 6 7 8 9 10 # простое перечисление  
do ping -c 1 192.168.10.$num  
done
```

```
for num in $(seq 1 10) # генерация из внешней команды  
do ping -c 1 192.168.10.$num  
done
```

```
for num in {1..10} # генерация встроенными средствами  
do ping -c 1 192.168.10.$num  
done
```

Циклы. While, until. Операторы break, continue.

- Применяются когда нужно повторять до выполнения какого-то события, когда количество элементов неизвестно заранее либо изменяется динамически:
 - окончание файла
 - получен последний аргумент командной строки
 - ввод текста пользователем
 - ожидание хоста в сети после перезагрузки.

IFS – internal field separator

Переменная, регулирующая разделение параметров (аргументов) на слова

- Используется:
 - во время раскрытия параметров командной строки перед выполнением
 - редактирование командной строки (удаление слова, Ctrl+W)
 - чтение ввода пользователя командной read
- Значение по умолчанию: <пробел> <табуляция> <перевод строки>

IFS + перенаправление из цикла

```
while IFS=":" read -r name pass uid guid comment home shell
do
    echo $name $uid $home $shell
done < /etc/passwd
```

Массивы

Переменные с множеством значений

Присвоение значений:

```
array=('first element' 'second element' 'third element')  
array=([3]='fourth element' [4]='fifth element')  
array[0]='first element'  
array[1]='second element'
```

Обращение к элементам массива:

```
echo ${array[1]}  
echo ${array[2]}  
echo ${array[*]}  
( IFS=$'\n' ; echo "${array[*]}" )
```

Трап. Перехват сигналов.

```
lockfile=./mylockfile
if ( set -o noclobber; echo "$$" > "$lockfile" ) 2> /dev/null
then
    trap 'echo Dont stop me now' INT
    trap 'rm -f "$lockfile"; exit $?' TERM EXIT
    while true ; do
        ls -ld ${lockfile}
        sleep 2
    done
else
    echo "Failed to acquire lockfile: $lockfile."
    echo "Held by $(cat $lockfile)"
fi
```

Trap. Обработка ошибок.

```
trap "echo Error ${LINENO} ${BASH_COMMAND}" ERR  
true  
uname -a  
test_test  
pass
```

Проверки кода

<https://www.shellcheck.net>

```
cat file.txt | while read line; do  
    echo $line  
done
```

Исправленная версия

<https://www.shellcheck.net>

```
#!/usr/bin/env bash
while read -r line; do
    echo "$line"
done < file.txt
```

Итоги вебинара

Проверка достижения целей

1.	Разобрали синтаксис Bash
2.	Научились писать скрипты на Bash

Как изучать дальше?

- Advanced Bash Scripting Guide (ABSG).
Название на русском: Искусство написания Bash скриптов.

**Заполните, пожалуйста,
опрос о занятии
по ссылке в чате**