

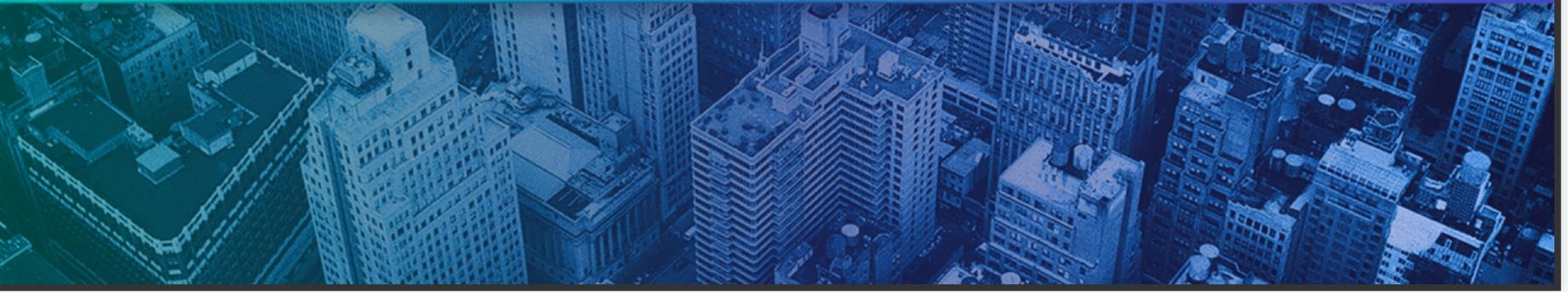


Онлайн-образование



Меня хорошо видно && слышно?

Ставьте  , если все хорошо
Напишите в чат, если есть проблемы



НЕ ЗАБЫТЬ ВКЛЮЧИТЬ
ЗАПИСЬ!!!

Оболочка BASH

После занятия вы сможете

1. Написать и запустить скрипт Bash
2. Использовать основные синтаксические конструкции
3. Читать скрипты

Зачем вам это уметь

ВАШ ВАРИАНТ?

Зачем вам это уметь

МОЙ ВАРИАНТ

1. Исправлять ошибки в скриптах
2. Повысить свою производительность
3. Убрать/автоматизировать рутинные задачи

План работы

- Назначение оболочки
- Запуск скриптов
- Подстановки (expansions)
- Перенаправление ввода вывода
- Условия
- Циклы

Что такое Unix shell?

- Обычная программа, запускающаяся после входа в систему
- Интерактивный командный интерпретатор
- Платформа интеграции для утилит (glue-language)
- Язык программирования (скрипты из списка команд)
- Макропроцессор (программа выполняющая преобразование текста)

Изучаем только Bash. Bourne Again SHeLL. Есть и другие zsh, fish

Команды

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

Выполнить примеры и определить тип команд

```
type type help alias read
type dmesg rm man
type if case
type -a ls
type -a echo pwd test
```

Скрипт

- Последовательность команд
- Разделители команд:
 - перевод строки или точка с запятой ;
 - | вертикальная черта (на следующем занятии)
 - && и ||

Первая строка : shebang

```
#!something или чём мы запускаем скрипт.  
#!/bin/sh # по умолчанию  
#!/bin/rm  
#!/bin/less  
#!/bin/bash  
#!/usr/bin/env bash
```

Варианты запуска:

Создаем файл script.sh

```
# вариант запуска 1
```

```
bash script.sh
```

```
# вариант запуска 2
```

```
chmod +x ./script.sh  
./script.sh
```

```
# вариант запуска 3
```

```
source script.sh
```

Подстановки текста (expansion)

- brace expansion { } фигурные скобки
- tilde expansion ~ (тильда)
- parameter and variable expansion **\$name \${name}**
- command substitution \$() круглые скобки
- arithmetic expansion (), \$() двойные круглые скобки
- word splitting " двойные кавычки, ' одинарные кавочки,
пробел
- filename expansion (globbing) *, ?, [] Оболочка выполняет
замену и подстановку текста в зависимости от
синтаксической конструкции

Примеры подстановки текста

```
echo {1..10}
echo ~
echo $PWD ${PATH}
echo $(uname) $(hostname) $(date)
echo $(( 2+ 2)) $(( 10*10))
echo /bin/???
```

Параметры vs переменные

Параметр хранит значение, получить которое можно обращаясь к параметру по

- **имени.** Другое название - **переменные**.
 - Примеры: company, USER, Year
- **числу.** Другое название - **позиционные параметры**.
 - Примеры: 1, 2, 3
- **спецсимволу.** Получают значение автоматически.
 - Примеры: ?, #, UID

Подстановка параметров

Подстановка значения выполняется если перед именем используется знак \$ или \${ }

- Примеры подстановки
 - **переменных:** \$company, \$USER, \$Year
 - **позиционных параметров:** \$1, \$2, \$3
 - **спецсимволов:** \$?, \$#, \$UID

Присваиваем значение переменной и обращаемся к нему

```
company=Otus; Year=2021 ; echo $company $Year
echo $year $0 $? # ничего не присваивали
echo ${company} ${Year} ${PID} ${0} ${?} # еще один формат
```

Какое имя переменной корректное?

```
courses=otus
10courses=otus
_courses=otus
courses10=otus
```

Команды чтобы получить список переменных

- env
- printenv
- declare
- export

Экранирование

Присвоить переменной motto значение: I love
OTUS!!!

```
motto=I love OTUS!!!
echo $motto
```

Результат выполнения команды

Shell return code, exit code выставляется автоматически:

- 0 - команда выполнена успешно
- 1..255 - команда выполнилась с ошибкой
- Код возврата доступен через переменную `$?`
- Позволяет организовывать условия в программе

```
ls /etc/; echo $? # return code 0
ls /abc/; echo $? # return code 1
```

```
exit 1 # выставление exit code из скрипта
return 1 # выставление exit code из функции
```

Условия

```
help if

if COMMANDS; then COMMANDS; fi

if COMMANDS; then COMMANDS; else COMMANDS; fi

if COMMANDS; then COMMANDS;
  elif COMMANDS;
    then COMMANDS;
    else COMMANDS;
fi
```

```
if grep vagrant /etc/passwd # любая команда которая
# выставляет exit code
then
  echo user vagrant in file /etc/passwd
fi
```

test выставляет exit code

И больше ничего не делает!

```
test -f /etc/ ; echo $? # существует ли файл?  
test -d /etc/ ; echo $? # существует ли директория?
```

Варианты реализации команды

```
type -a test [ [ [
```

Что умеет сравнивать команда?

```
help test
```

Строки, файлы, числа

Пример использования условного оператора

```
# info.sh
script [help | disk | memory]
help - shows help message
disk - shows disk usage
memory - shows memory usage
```

Пользователь передает один из параметров. В зависимости от аргумента - принимаем решение какую команду выполнять.

Позиционные параметры

Передаются при запуске скрипта, средство передачи данных в скрипт.

```
# ./myscript.sh  
echo '$1:' $1  
echo '$2:' $2
```

```
./myscript.sh file.txt /tmp/
```

- \$0-9 -- значение соответствующего параметра (а если больше 9 ?)
- \$# -- количество переданных параметров
- \$* -- представляется, как одна строка

Пример реализации

```
#!/usr/bin/env bash
arg="$1"
if [ "$arg" == help ] ; then
    echo 'script [help | disk | memory]'
    echo 'help - shows help message'
    echo 'disk - shows disk usage'
    echo 'memory - shows memory usage'
fi

if [ "$arg" == disk ] ; then
    df
fi
```

Перенаправление ввода вывода

```
ls -l /dev/std*
```

Перенаправление **stdin FD=0**

```
command < file  
command <&0 file
```

Перенаправление **stderr FD=2**

```
command1 2>&1 | command2  
command 1>file 2>&1  
command 2>file 1>&2
```

Here string, Here docs

HERE STRING

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

HERE DOCS

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

Используем here doc

```
#!/usr/bin/env bash
arg="$1"
if [ "$arg" == help ] ; then
cat <<END
script [help | disk | memory]
  help - shows help message
  disk - shows disk usage
  memory - shows memory usage
END
fi
```

Циклы for

```
help for
```

Циклы for. Последовательность строк.

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

Действие над файлами.

```
for file in *
    do md5sum $file
done
```

Циклы for. Цифровая последовательность.

```
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
```

```
for ((num=1;i<11;num++)) # Арифметический формат (C-like)
do ping -c 1 192.168.10.$num
done
```

Циклы while, until

```
help while  
help until  
help break  
help continue
```

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

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>**

Перенаправление из цикла.

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

Построчное чтение из файла.

```
while IFS= read -r line; do  
printf '%s\n' "$line"  
done < "$file"
```

Цикл until. Пример.

Ожидаем хост после перезагрузки.

```
until ping -q -c 3 $host 1>/dev/null 2>&1 && nc -z $host 22
do
    sleep 1
    echo unavailable;
done
```

Массивы синтаксис

Присвоение

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

Обращаемся к элементам массива

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

Массивы пример

```
unset cmdss
cmdss=( $(echo /bin/???) )
echo ${cmdss[1]}
echo ${cmdss[0]}
( IFS=$'\n' ; echo "${files[*]}" )
```

Средства разработки

- Integrated development environment (vim, emacs, VSCode, Pycharm, Atom)
 - подсветка синтаксиса
 - проверка синтаксиса
 - автоматическое форматирование
- автоматическое форматирование кода
 - shfmt <https://github.com/mvdan/sh#shfmt>
- Ищем "дурнопахнущие" места в коде
 - <https://www.shellcheck.net/>
 - <https://mywiki.wooledge.org/BashPitfalls>
 - <https://mywiki.wooledge.org/BashGuide/Practices>

Проверки кода

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

<https://www.shellcheck.net/>

Результат

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

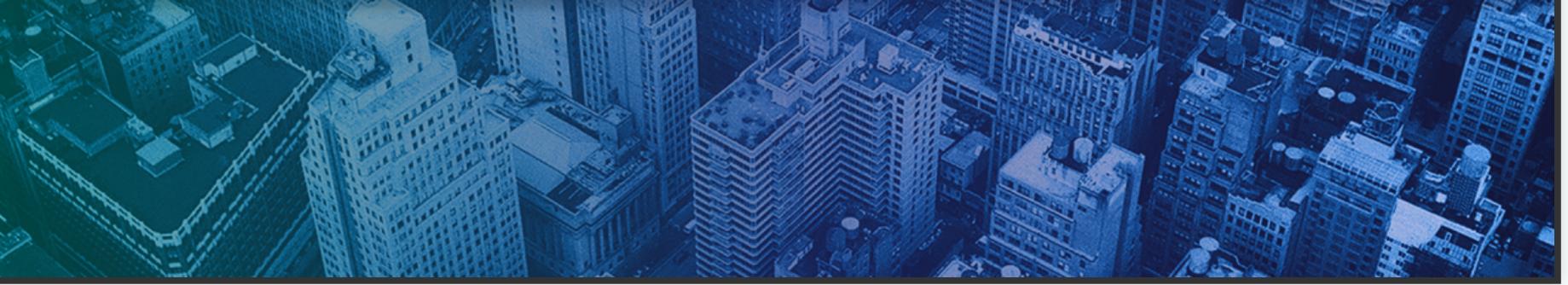
Рефлексия



Отметьте 3 пункта, которые вам запомнились с вебинара



Что вы будете применять в работе из сегодняшнего вебинара?



Заполните, пожалуйста,
опрос о занятии по ссылке в чате