

Если вы видите что-то необычное, просто сообщите мне.

# Devops

- Что такое CI\CD
- Что такое Devops
- Linux\Bash Работа в командной строке.

# Что такое CI\CD

CI\CD содержит три основных момента:

- Continuous Integration (CI)
- Continuous Delivery (CD)
- Continuous Deployment (CD)

## Этапы

- Написание кода.
- Сборка.
- Ручное/Автоматическое тестирование.
- Релиз.
- Развертывание.
- Поддержка и мониторинг.
- Планирование.

## Цели

- Сегрегация ответственности
- Снижение риска
- Короткий цикл обратной связи.

## Состав команды.

- Разработчики и дизайнеры (Dev)
- Инженеры по качеству (QA)

- Бизнес-аналитики (BAs) и владельцы продуктов (POs)
- Оперативный отдел (Ops)/ DevOps-инженеры
- Пользователи

# Инструменты

- GitLab, Github.
- Docker, Kubernetes
- Travis-CI, Circle-CI, Jenkins, TeamCity

# Что такое Devops

DevOps — это культура, которая способствует сотрудничеству между группой разработки и эксплуатации для более быстрого и автоматизированного развертывания кода в производстве. Слово «DevOps» представляет собой сочетание двух слов «разработка» и «операции».

DevOps помогает повысить скорость организации для доставки приложений и услуг. Это позволяет организациям лучше обслуживать своих клиентов и более активно конкурировать на рынке.

Проще говоря, DevOps можно определить как согласование процессов разработки и ИТ с улучшением взаимодействия и совместной работы.

## Почему DevOps используется?

DevOps позволяет командам Agile Development реализовывать непрерывную интеграцию и непрерывную доставку. Это помогает им быстрее выводить продукты на рынок.

Другие важные причины:

1. Предсказуемость
2. Воспроизводимость
3. Ремонтопригодность
4. Время выхода на рынок
5. Повышенное качество
6. Снижение риска
7. Отказоустойчивость
8. Экономическая эффективность

9. Разбивает большую кодовую базу на маленькие кусочки

# Жизненный цикл DevOps

1. Разработка
2. Тестирование
3. Интеграция
4. Развертывание
5. Мониторинг

## Цели DevOps

Вот шесть принципов, которые необходимы при принятии DevOps:

1. Ориентация на клиента
2. Непрерывная ответственность
3. Постоянное улучшение
4. Автоматизируйте все
5. Работайте одной командой
6. Контролируйте и тестируйте все

## Кто такой инженер DevOps?

DevOps Engineer — это ИТ-специалист, который работает с разработчиками программного обеспечения, системными операторами и другими производственными ИТ-специалистами для администрирования выпусков кода. DevOps должен обладать как сложными, так и мягкими навыками общения и совместной работы с командами разработчиков, тестирования и эксплуатации.

Подход DevOps требует частых, постепенных изменений версий кода, что означает частые схемы развертывания и тестирования. Хотя инженерам DevOps нужно время от времени писать код с нуля, важно, чтобы они имели основы языков разработки программного обеспечения.

Инженер DevOps будет работать с персоналом команды разработчиков, чтобы заняться кодированием и сценариями, необходимыми для соединения элементов кода, таких как библиотеки или наборы для разработки программного обеспечения.

# Роли, обязанности и навыки инженера DevOps

Инженеры DevOps работают полный рабочий день. Они несут ответственность за производство и текущее обслуживание платформы программного приложения.

Ниже приведены некоторые ожидаемые роли, обязанности и навыки, которые ожидаются от инженера DevOps:

- Способен выполнять поиск и устранение неисправностей системы в разных областях платформы и приложений.
- Эффективное управление проектом через открытые, основанные на стандартах платформы
- Повысить видимость проекта и прослеживаемость
- Улучшение качества и снижение затрат на разработку с помощью совместной работы
- Анализировать, проектировать и оценивать скрипты и системы автоматизации
- Обеспечение критического разрешения системных проблем с использованием лучших сервисов облачной безопасности.
- Инженер DevOps должен обладать умением решать проблемы и быстро учиться

# Linux\Bash Работа в командной строке.

## Права доступа к файлам

У файла есть три группы владельцев:

- User(пользователь)
- Group(группа)
- Other(все другие)

Каждой группе назначается права доступа:

- Read(Чтение)
- Write(Запись)
- Execute(Выполнение)

Рассмотрим пример Выполните в командной строке:

```
ls -l
```

В ответ мы получим вывод следующего вида:

```
-rw-r--r--. 1 user user    478 сен  6 16:42 index.html
-rwxr-xr-x. 1 user user   1244 июл 30 10:37 modules
drwxr-xr-x. 5 user user   4096 сен 10 15:58 Pictures
drwxr-xr-x. 2 user user   4096 янв 24  2021 Public
drwxr-xr-x. 13 user user  4096 авг 17 09:12 Repos
```

Разберем запись:

```
drwxr-xr--
```

Нотация имеет группы свойств:

- `d` или `-` укзание того, что перед нами файл или папка.
- `rwX` - позволяет пользователю читать(`r`), писать(`w`), исполнять(`x`) этот файл
- `r-x` - позволяет группе пользователей читать(`r`), но не писать в него (`-`), исполнять(`x`)
- `r-x` - позволяет всем остальным читать(`r`), но не писать в него (`-`), не исполнять(`-`)

`---` - подразумевает что у владельца, группы или остальных нет никаких разрешений вообще.

# CHMOD

Довольно часто для смены прав доступа используется команда `chmod`.

```
chmod permissions filename
```

Используя табилцу ниже файлу или папке можно дать различные права:

Ч	Т	С
и	и	и
с	п	м
л	р	в
о	а	о
	з	л
	р	
	е	
	ш	
	е	
	н	
	и	
	я	



0	О тс у тс т в у е т р а з р е ш е н и е н а	---
1	В ы п о л н е н и е	-- х
2	З а п и сь	- w -

3	В ы п о л н е н и е + З а п и сь	- w х
4	Ч т е н и е	r--
5	Ч т е н и е + В ы п о л н е н и е	r- х

6	Ч т е н и е + З а п и сь	r w -
7	Ч т е н и е + З а п и сь + В Ы п о л н е н и е	r w x

# CHOWN

Команда для изменения владельца файла\каталога

```
chown user:group filename
```

Возьмем пример выше

```
-rwxr-xr-x. 1 user user    1244 июл 30 10:37 modules
```

Если мы воспользуемся командой `chown`:

```
chown test:group modules
```

То в результате `user` должен смениться на `test`, и `user` на `group`, и при проверке прав мы увидим следующие изменения:

```
-rwxr-xr-x. 1 test group    1244 июл 30 10:37 modules
```

# Перенаправление ввода

">"

Этот символ используется для перенаправления вывода(stdout)

Пример:

```
la -al > file.txt
```

В этом случае всё что выведется командой `la -al` будет записано в `file.txt`, при этом на экране ничего не отобразится.

Можно рассмотреть следующий пример:

```
cat music.mp3 > /dev/audio
```

Команда `cat` прочитает файл `music.mp3` и вместо того чтобы вывести его в терминал отправит в устройство `/dev/audio`, если у вас в системе есть такое устройство то вы услышите этот файл из колонок.

"<"

Этот символ используется для перенаправления ввода(stdin)

Пример:

```
mail -s "subject" to-address < Filename
```

Команда прикрепит `Filename` к письму и отправит его по адресу `to-address`.

# Перенаправление ошибок

Всякий раз когда выполняется программа открыто 3 файла:

- standard input -- FD0
- standard output -- FD1
- standard error -- FD2

FD - файловый дескриптор. В Linux / Unix все является файлом. Обычный файл, каталоги и даже устройства являются файлами. Каждый файл имеет связанный номер, называемый дескриптором файла (FD).

примеры использования:

1. Вывод ошибок программы `myprogram` перенаправляется в `errorfile`. В этом случае ошибки не отображаются при выполнении команды `myprogram`.

```
myprogram 2>errorfile
```

2. С помощью `find` мы ищем в текущем каталоге `.` файл начинающийся с `my`. При этом все ошибки описки отправляются в файл `error.log`

```
find . -name 'my*' 2>error.log
```

3. Команда выводит список файлов в двух директориях `Documents` и `ABC`, в свою очередь `2>&1` переводит вывод ошибок в стандартный вывод. То есть `stderr` перенаправляется в `stdout`. И весь вывод записывается в файл `dirlist`

```
ls Documents ABC> dirlist 2>&1
```

# Pipeline

Pipeline - комбинация команд передающая аргументы друг другу после выполнения. Т.е. вывод команды является вводом следующей команды

Например, ниже мы передаем вывод команды `cat` на вход команды `less`:

```
cat filename | less
```

## grep

Создадим файл `example.txt`, внесем в него какие-либо строки и выведем его:

```
cat example.txt
```

Команда `grep` выведет искомые данные. Синтаксис использования: `grep ИСКОМОЕ_ВЫРАЖЕНИЕ`:

```
cat example.txt | grep dog
```

Команда `grep` пройдет по выводу команды `cat` и отразит только строки содержащие `dog`.

Ключи команды `grep`:

к л ю ч ь	ф у н к ц и я
-----------------------	---------------------------------

-v	Показывает все строки, которые не соответствуют искомой строке
-с	Отображает только количество совпадающих строк

-n	Показывает совпадающую строку и ее номер
-l	Показывает только имя файла со строкой

# sort

Сортирует содержимое в алфавитном порядке

Команда `sort` выведет искомые данные. Синтаксис использования: `sort ФАЙЛ`.

```
sort example.txt
```

Ниже приведен вариант использования `sort` в комбинации с командой `grep` в pipeline:

```
cat example.txt | grep -v a | sort -r
```



`grep` выведет все строки не содержащие букву `a`, а затем отсортирует их в обратном алфавитном порядке.

# Регулярные выражения

Регулярные выражения — это специальные символы, которые помогают искать данные, соответствующие сложным шаблонам. Регулярные выражения сокращаются до `'regex'` или `'regex'`.

## Основные Регулярные выражения

Символ	Описание
,	заменяет любой символ
^	соответствует началу строки
\$	соответствует концу строки

Сим вол	Опи сан ия
*	соо тве тств ует нул ю или бол ее раз пре дыд уще го сим вол а
\	Пре дст авл яют спе циа льн ые сим вол ы
()	Гру ппы рег уля рны х выр аже ний

Символ	Описание
?	Соответствует ровно одному символу

Интервал в регулярных выражениях

Выражение	Описание
{n}	Соответствует предыдущему символу, появляющемуся точно n раз

Вы ра же ние	Опи сан ие
{n, m}	Соо тве тств ует пре дыд уще му сим вол у, поя вля ющ ему ся <div>n</div> раз, но не бол ее чем <div>m</div>

Вы ра же ние	Опи сан ие
{n,}	Соо тве тств ует пре дыд уще му сим вол у, тол ько есл и он поя вля етс я <b>n</b> раз или бол ее

Примеры использования:

Создадим файл `example.txt` и внесем в него какой-то текст

Чтобы посмотреть содержимое выполним:

```
cat example.txt
```

Выведет все строки содержащие букву **a**

```
cat example.txt | grep a
```

Выведет все строки начинающиеся с буквы **a**

```
cat example.txt | grep ^a
```

Выведет все строки заканчивающиеся на `t`

```
cat example.txt | grep t$
```

Выведет строки где символ `p` идет 2 раза подряд

```
cat example.txt | grep -E p{2}
```