Урок 4

**Текстовые редакторы, немного о правах и ключи ssh.**

Итак, текстовые редакторы. Как мы помним, все файлы в ОС Linux являются текстовыми, а это значит, что одна из первых программ для удобной работы была текстовым редактором. История текстовых редакторов уходит в семидесятые, когда появился Vi - или виай по-русски.

Чем он крут?

**Редактор vi** – один из старейших и мощнейших редакторов в Linux. Он работает даже на таких аппаратных терминалах как VT100 и XTERM. Современный вариант этого редактора называется vim (Vi IMproved – улучшенный vi). vim эмулирует все команды vi,и в то же время добавляет много дополнительных возможностей, таких как:

* - многоуровневый процесс отмены ранее выполненных действий ( undo );
* - использование нескольких окон редактирования;
* - редактирование командной строки;
* - встроенная справочная система (команда :help ) и многое другое.

Сейчас не найдется ни одного дистрибутива, чтобы там не было редактора Vi. Ведь рано или поздно пользователь столкнется с проблемой, чтобы зайти в текстовый файл и сделать там правки.

Шпаргалка:

* **ZZ** (Note: capitals) - Save and exit
* **:q!** - discard all changes, since the last save, and exit
* **:w** - save file but don't exit
* **:wq** - again, save and exit
* :x - again, save and exit
* **G** (shift g) - move to the last line
* / - search (n - next)
* i - insert text, edit mode. ECS to exit
* u - undo last action
* **dd** - delete the current line ( i mode not needed)
* Del - delete single character ( i mode not needed)

Давайте создадим файл.

В линуксе, при создании файлов случается так, что мы указываем, что хотим создать файл, указываем, каким редактором будем пользоваться, но файл будет создан и сохранен только когда мы выйдем из редактора с сохранением.

**vi /tmp/file**

И перед нами - интерфейс! Представьте, как раньше (а теперь и мы), страдали. Вот так он страшно выглядит. Но только на первый взгляд.

В сравнении с каким-нибудь MS Office полное дно.

Кстати! Забыл сказать, что такое Vi - Vi - это visual! Визуальный, блин.

Что же мы видим. Пустой экран редактора. Он занимает всю площадь нашего экрана. И тут есть тильды. Они видны только нам и указывают на то, что это пустые строки. На самом деле тильд нету и если мы вдруг захотим распечатать что-то, то там будет пусто.

А в самом низу мы видим имя и путь к файлу и что курсор находится на первой строке из всех строк и это 100% от всего документа. То Есть документ новый и чистый.

А теперь давайте присмотримся к левому нижнему углу. Особо зоркие могут заметить еще один символ, который как и кавычки в Linux, можно спутать с другим. Это не тильда, это минус.

Он нам говорит, что мы находимся в командном режиме (и о боже, в нашем интерфейсе). Еще некоторые его называют режимом просмотра.

Когда мы видим такой символ, это означает, что Vi ждет от нас команд, либо мы просто зашли сюда поглазеть на документ.

Но что же делать, если мы хотим что-то дописать? И тут начинаются очередные сложности, к которым я думаю вы уже привыкаете, используя Linux.

**Нам необходимо перейти в другой режим.**

Перейти в другой режим мы можем клавишей I. Теперь присмотримся к нашему левому нижнему углу. Что видим? Букву i. Это сокращение от insert. Значит, мы находимся в режиме ввода и редактирования. И мы можем написать какой-то текст:

hello from tel ran

end

~

мы можем надобавлять каких-либо строчек и написать что угодно.

Стоит сказать, что бывают и русскоязычные дистрибутивы, где вместо буквы I будет написано ВСТАВКА. Выглядит забавно.

В более продвинутых версиях может быть написано Insert.

Теперь давайте попробуем выйти из редактора с сохранением. Как видно в подсказке, это две заглавные буквы ZZ. Иногда в терминале это может не срабатывать с первого раза, как у меня на прошлом занятии. Но ничего.

Кстати, вопрос “Как выйти из редактора Vi” с сохранением или без, является одним из самых топовых в среде Linux до сих пор на протяжении многих десятков лет.

А чтобы прочесть один из самых простых мануалов по среде Vi, нам понадобится куча времени, ведь книжка занимает 570 страниц.

Но нам, как пользователям, понадобится лишь наша шпаргалка, которая удовлетворяет потребности в работе.

Едем дальше.

Для того, чтобы сохранить и выйти (надеюсь терминал не будет адово тормозить из моей страны), нам нужно выйти из режима редактирования. Давайте еще раз присмотримся к левому нижнему углу. Там буква I. Теперь нажмем на **esc**.

Теперь буква i заменилась на тире. А это означает, что мы вышли из режима ввода в режим команд (или режим просмотра).

Теперь нужно выйти непосредственно из редактора. С сохранением. Я покажу вам целых три способа для выхода. Но вам все три не нужны, главное запомнить один, который больше понравится.

:x :wq ZZ

Это три варианта выхода с сохранением.

Воспользуемся одним из способов. И что мы видим? Что из Vi мы благополучно вышли и система ждет наших команд.

Вы можете спросить, а если мне не нужно делать сохранение? Что же делать и как быть? Для этого нужно нажать:

:q!

Едем дальше.

Теперь мы можем посмотреть чуть глубже в режим команд и давайте я покажу, как в этом режиме удалять строки.

Все просто, для этого есть команда **dd**.

Но если вдруг вы удалили лишнего - есть клавиша **U - undo**.

Давайте выйдем из редактора и нажмем ctrl+l (или команду clear), чтобы не видеть остатки от редактора в нашем терминале.

Давайте проверим че мы там поназаписывали:

cat /tmp/file

**Это был Vi. Он есть везде.**

А теперь давайте шагнем вперед на целых 20 лет и посмотрим, что же разработчики выдали хорошего.

Редактор Nano.

Единственным минусом, который можно назвать, является его отсутствие на некоторых дистрибутивах. Несмотря на очень маленький размер - около 20 килобайт, некоторые разработчики, делая дистрибутив экономят даже на этом. И не включают этот редактор в стандартный пакет.

Давайте запустим:

nano /tmp/file

И что мы видим? Интерфейс, который уже хоть на что-то похож.

Сверху мы видим название файла. Уже прекрасно. Мы сразу находимся в режиме редактирования и можем дописать, отписать, убрать и прочее.

И никаких тильд в пустом пространстве, которые бесят и пытаются сбить с толку.

Дальше, если мы вдруг забыли как нам отсюда выходить или что-то сделать, то тут мы сможем увидеть подсказки, которых нам так не хватало в Vi.

Вот, смотрите, чтобы выйти нужно нажать ctrl+x. Вот эта крышечка - это ctrl.

Давайте введем текст и попробуем выйти.

Смотрите - 20 лет прошли не зря!

У нас при выходе спрашивают, хотим ли мы сохранить файл и далее, хотим ли мы переименовать наш файл. Магия!

Как по мне, так более хороший подход и пользоваться попроще.

Теперь давайте внесем изменения в наш файл при помощи nano.

nano /tmp/file

У всех получилось?

Давайте сохраним и выйдем.

Супер, двигаемся дальше.

Вы уже наверняка что-то программировали и знаете что такое переменные.

Переменная простыми словами — это хранилище данных. Сюда можно положить какое-то значение (например, число, строку или другой тип данных).

Давайте поговорим, как в ОС Linux работают переменные. Есть много интересных моментов.

В Linux есть одна переменная, которая влияет на запуск всех программ в системе.

В этой ОС переменные принято (это не значит, что не заработает, это очередное джентльменское соглашение), писать большими буквами (капсом).

Давайте для начала посмотрим на команду **env,** которая вернет нам все объявленные переменные в нашей ОС.

localhost:~/test1# env

SHLVL=3

HOME=/root

OLDPWD=/root

PAGER=less

PS1=\h:\w\$

TERM=linux

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

PWD=/root/test1

TZ=UTC-03:00

Тут мы видим имя переменной слева и ее значение после равно справа.

Эти переменные заведены ОС. Но мы можем сделать собственные. Давайте попробуем:

делается это при помощи команды:

export MYVAR=hello

export NAME=Ivann

Каждый может вставить свое имя.

Теперь, если мы сделаем команду env, то мы сможем увидеть, что эти переменные попали в нашу систему:

localhost:~/test1# env

SHLVL=3

HOME=/root

OLDPWD=/root

PAGER=less

PS1=\h:\w\$

**NAME=Ivan**

**TERM=linux**

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

MYVAR=hello

PWD=/root/test1

TZ=UTC-03:00

Как в системе их использовать в дальнейшем? В ОС Linux вызов переменной происходит через знак $

Например мы берем команду **echo** (напомните, что она нам возвращает?)

echo $MYVAR $NAME

hello Ivan

И нам вернулось на экран Hello Ivan.

Видите, мы объявили переменные, а потом вызвали их и вместо них увидели слова.

То Есть при помощи echo мы напечатаем значения этих двух переменных.

Для чего нам это может понадобиться?

В будущем, когда мы будем запускать всякие скрипты, где в сценарии с каждой новой строкой будет написано новое действие, нам не нужно будет писать одни и те же данные каждый раз. Достаточно один раз сделать переменную и указать ей, что нужно показывать. Например, адрес, параметр или чего душе угодно. Эта штука здорово экономит нам время при написании скриптов.

Например, есть скрипт, который лезет в базу данных, там берет какие-то данные и складывает их в другую базу данных.

Так вот точки доступа (адрес базы данных, имя пользователя, пароли и т.д.) можно сделать при помощи переменной и упростить себе работу.

**Теперь важная переменная от которой много чего зависит. PATH**

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

PATH - путь.

Вы наверняка заметили, что когда мы запускаем команду, то она запускается из любого места в системе.

например команда **ls** выполнилась из любого места.

Как это работает и какая логика?

Сначала система пытается найти файл в папке (где мы находимся) с именем нашей команды. Но не находит. Затем ОС обращается к переменной PATH, где перечислены возможные локации,где может лежать исполняемый файл.

Причем эти локации перечислены через двоеточие.

PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Логически доходит до /usr/bin и там находит исполняемый файл для команды ls.

Как это может нам пригодиться? Например, мы написали программу vasya. И хотим ее запускать, как мы запускаем редактор nano. Прямо из командной строки. Но когда мы вводим название программы - мы видим, что система не понимает, что и откуда нужно запускать. Тут нам и поможет наш PATH. Мы делаем еще один путь, где ОС будет искать путь исполнения для нашей программы vasya. И запускать ее оттуда. Более того, как только мы задействуем PATH, то будем использовать и ТАБ для поиска и подсказки в названии нашей программы (как мы пользуемся подсказками по командам и папкам).

С этим, вы, возможно, столкнетесь.

А если пойдете на тестировщика, то столкнетесь 100%.

Двигаемся дальше.

Я обещал рассказать про наши права и что мы видим, когда делаем команду **ls -la**

давайте рассмотрим на папке /tmp

localhost:~/test1# ls -la /tmp

total 12

drwxrwxrwt 2 root root 58 Jan 9 2021 .

drwxrwxrwx 21 root root 461 Feb 9 16:04 ..

-rw-r--r-- 1 root root 15 Feb 9 15:21 file

d

Означает, что это директория.

Если тире - значит файл.

Бывает l - линк

всего девять символов.

Первые три - права владельца файлов.

Вторые три - права группы владельцев

И последние три - все остальные.

Рассмотрим подробнее, что значат условные значения флагов прав:

* **---** - нет прав, совсем;
* **--x** - разрешено только выполнение файла, как программы но не изменение и не чтение;
* **-w-** - разрешена только запись и изменение файла;
* **-wx** - разрешено изменение и выполнение, но в случае с каталогом, вы не можете посмотреть его содержимое;
* **r--** - права только на чтение;
* **r-x** - только чтение и выполнение, без права на запись;
* **rw-** - права на чтение и запись, но без выполнения;
* **rwx** - все права;
* **--s** - установлен SUID или SGID бит, первый отображается в поле для владельца, второй для группы;
* **--t** - установлен sticky-bit, а значит пользователи не могут удалить этот файл.

В нашем варианте суперадмин и группа пользователя совпадают по названию, но так бывает далеко не всегда. Давайте приведу пример своего терминала.

SSH-ключ **Secure Shell (безопасная оболочка)**— это специальный код, который позволяет удалённому компьютеру понять кто вы есть и какими правами на этом компьютере обладаете.

## **Как работает SSH-ключ**

SSH-ключ разделён на две части. Одна часть называется приватной и должна всегда храниться только на вашем компьютере. Вторая часть ключа называется публичной и эту часть нужно копировать на другие компьютеры. При подключении к удалённому компьютеру он сравнивает публичную часть, которую вы ему дали с приватной частью, которая хранится у вас. Если части ключа совпадают, то вы получаете доступ к удалённому компьютеру (на самом деле всё гораздо сложнее, но принцип работы именно такой). На вашем компьютере может быть создано сколько угодно SSH-ключей. То есть вы можете использовать один SSH-ключ для доступа к сотне компьютеров или для каждого удалённого компьютера создавать отдельный ключ.

## **Защита SSH ключа**

Вы никогда и ни при каких обстоятельствах не должны никому передавать приватную часть своего ключа. Чтобы эту приватную часть нельзя было украсть можно её зашифровать, тогда даже если она попадёт в чужие руки её не смогут использовать. Если вы зашифруете приватную часть SSH-ключа, то каждый раз при его использовании вам нужно будет вводить ключ шифрования, называемый passphrase. Есть общее правило безопасности: если ключ используется не скриптом, а человеком, то он должен быть зашифрован.

## **Как создать SSH-ключ под Windows**

Windows не имеет встроенной поддержки SSH протокола. Поэтому, для создания SSH-ключа нужна специальная программа. Подойдёт Git Bash, которая [устанавливается вместе с Git под Windows](https://blog.vistro.ru/ide/git/how-to-install-git-on-windows/).

Инструкция по созданию SSH-ключа:

1. запустите Git Bash (используется только в ОС Windows, на Linux & MacOS есть свои терминалы);
2. введите команду ssh-keygen;
3. программа предложит выбрать место, куда сохранить ключ. По умолчанию создаётся каталог .ssh в папке текущего пользователя Windows. Например, у меня это C:\Users\Denis\.ssh. Этот вариант подходит, поэтому нажимайте Enter;
4. программа запросит passphrase для шифрования ключа. Если просто нажать Enter, то приватная часть ключа не будет зашифрована. Помните, что если SSH-ключ используется не скриптом, то он должен быть зашифрован. Относитесь к passphrase как к паролю, его нужно запомнить;
5. после нажатия Enter, программа попросит ввести passphrase повторно. Если вы не шифруете ключ, то ничего не вводите. Следующее нажатие Enter завершит создание ключа.

Теперь в каталоге .ssh созданы два файла:

* id\_rsa — приватный ключ (никому не передавать);
* id\_rsa.pub — публичный ключ (нужно копировать на другие компьютеры).

Git — система управления версиями с распределенной архитектурой. В отличие от некогда популярных систем вроде CVS и Subversion (SVN), где полная история версий проекта доступна лишь в одном месте, в Git каждая рабочая копия кода сама по себе является репозиторием. Это позволяет всем разработчикам хранить историю изменений в полном объеме.

Подсказки по уроку:

**Статья про ГИТ:**

<https://habr.com/ru/post/541258/>

**генерируем ключ SSH:**

ssh-keygen

**Посмотреть доступ к файлам с ключами:**

ls -la .ssh

**Имя сервера:**

ssh -i ~/.ssh/id\_rsa [ec2-user@linux.telran-edu.de](mailto:ec2-user@linux.telran-edu.de)

**Добавление ключа редактором nano:**

nano .ssh/authorized\_keys

**показать ключ:**

cat .ssh/id\_rsa.pub

**Выйти из nano:**

ctrl+x