

# Файловые системы



Сергей  
Андрюнин



**Сергей Андрюнин**

DevOps инженер, RTLabs

---

# План модуля

1. Работа в терминале, лекция 1
2. Работа в терминале, лекция 2
3. Операционные системы, лекция 1
4. Операционные системы, лекция 2
5. **Файловые системы**
6. Компьютерные сети, лекция 1
7. Компьютерные сети, лекция 2
8. Компьютерные сети, лекция 3
9. Элементы безопасности информационных систем



# План занятия

1. [Объекты файловой системы](#)
2. [Права доступа](#)
3. [Организация хранения данных в Linux](#)
4. [RAID/mdadm](#)
5. [LVM2](#)
6. [Таблица разделов](#)
7. [Создание и монтирование файловых систем](#)
8. [Итоги](#)
9. [Домашнее задание](#)



# Объекты файловой системы

## man 2 stat, /stat \{

Что такое файл? Некоторые важные поля объектов файловой системы:

```
struct stat {
    dev_t      st_dev;      /* ID устройства, содержащего файл */
    ino_t      st_ino;      /* номер Inode */
    mode_t     st_mode;     /* биты доступа + тип файла */
    nlink_t    st_nlink;    /* число hard links (жестких ссылок) */
    uid_t      st_uid;     /* User ID - принадлежность пользователю */
    gid_t      st_gid;     /* Group ID - принадлежность группе */
    dev_t      st_rdev;
    /* Device ID - тип устройства, если файл является
    файлом специального назначения */
    off_t      st_size;     /* Размер в байтах */
    struct timespec st_atim; /* Время последнего доступа */
    struct timespec st_mtim; /* Время последнего изменения содержимого */
    struct timespec st_ctim; /* Время последнего изменения метаданных */
    ...
}
```

Уже по структуре данных системного вызова **stat** мы видим, что файл это не обязательно область на диске с данными. Например, файл может представлять устройство.

*timespec* – не всегда времена доступа честно обновляются из соображений производительности (опции монтирования).

# Команда stat

```
vagrant@netology1:~$ stat /etc/hosts
  File: /etc/hosts
  Size: 198          Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 131207       Links: 1
Access: (0644/-rw-r--r--)  Uid: (   0/   root)   Gid: (   0/   root)
Access: xxx 19:51:20.402137404 +0000
Modify: yyy 15:29:25.128599631 +0000
Change: zzz 15:29:25.128599631 +0000
```

- важнейшее поле – **Inode** (индексный дескриптор)
- тип файла: regular file (обычный файл)
- 1 жесткая ссылка на объект (поле **Links**)
- Uid – принадлежность пользователю, Gid – принадлежность группе

IO Block – размер блока на файловой системе

```
root@netology1:~# tune2fs -l /dev/mapper/vgvgagrant-root | grep 'Block size'
Block size:                4096
```

8 Blocks – размер физического блока устройства:

```
root@netology1:~# cat /sys/block/sda/queue/physical_block_size
512
```

Поэтому, даже при малом размере файла в 198 байт на устройстве выделено 8 блоков по 512 байт (так как минимальный блок ФС – 4096 байт).

# inode – индексный дескриптор

Уникальный в рамках файловой системы контейнер метаданных файла (пара inode+device – однозначный идентификатор объекта в системе).

stat читает данные из inode.

**inode** – первичный идентификатор объекта на файловой системе.

Имя таковым не является: в рамках одной ФС может быть создано более одного файла с одним и тем же inode и разными именами: число таких ссылок и показывает stat в поле Links. У обычного файла будет только одна ссылка. Создадим более одной ссылки:

```
root@netology1:~# touch test_file
root@netology1:~# stat --format=%h test_file
1
root@netology1:~# ln test_file this_is_hardlink_for_test_file
root@netology1:~# stat --format=%h test_file
2
root@netology1:~# stat --format=%i test_file; stat --format=%i
this_is_hardlink_for_test_file
1179657
1179657
```

Если мы удалим оригинальный test\_file, объект продолжит существование на ФС, так как на него будет присутствовать ссылка this\_is\_hardlink\_for\_test\_file:

```
root@netology1:~# rm test_file
root@netology1:~# stat --format=%h,%i this_is_hardlink_for_test_file
1,1179657
```



# inode, имена файлов

- Имена файлов и соответствие имен<>inode хранятся в структуре директории, в которой эти файлы находятся.
- Число inode определяет число объектов, которое может быть создано на ФС.
- Есть ФС, где inode предварительно выделяются при создании (семейство **ext**); есть те, где inode может быть добавлены динамически (**xfs**). Об этом важно знать, так как может сложиться ситуация, при которой на ФС есть свободное пространство для создания файлов, но уже нет inode (слишком много файлов).

```
root@netology1:~# df -h | grep root
/dev/mapper/vgvgagrant-root 62G  3.5G  55G   6% /
root@netology1:~# df -i | egrep '(Inodes|root)'
Filesystem                Inodes    IUsed    IFree   IUse%    Mounted on
/dev/mapper/vgvgagrant-root 4104192   100446  4003746    3%      /
```

Одна из встречающихся техник программирования в Linux – изменения поведения программы в зависимости от того, под каким именем ее вызвали.

```
root@netology1:~# ls -li /usr/bin/bzip2
3407895 -rwxr-xr-x 3 root root 39144 Sep  5  2019 /usr/bin/bzip2
root@netology1:~# find /usr/bin -inum 3407895
/usr/bin/bunzip2
/usr/bin/bzip2
/usr/bin/bzcat
```



В каких объектах файловой системы присутствует больше одной жесткой ссылки?

## dentry (directory entry), директории

Мы узнали, что название файла – запись об именовании inode в структуре, которая представляет директорию.

Эта структура называется **dentry**, когда мы просматриваем содержимое директорий `ls`, происходит системный вызов **getdents64**:

```
root@netology1:~# strace -e getdents64 ls  
getdents64(3, /* 4 entries */, 32768) = 112
```

Так как у любой директории помимо своего имени присутствует и короткая ссылка (`.`), у директорий всегда не менее двух `hardlink`.

# man 2 stat, /S\_IFMT

```
case S_IFREG:  printf("regular file\n");          break;
case S_IFDIR:  printf("directory\n");             break;
case S_IFLNK:  printf("symlink\n");               break;
case S_IFIFO:  printf("FIFO/pipe\n");             break;
case S_IFSOCK: printf("socket\n");                break;
case S_IFBLK:  printf("block device\n");          break;
case S_IFCHR:  printf("character device\n");      break;
```

В **man** по системному вызову **stat** описаны типы файлов, которые встречаются в Linux:

- **regular file**, стандартный файл;
- **directory**, директория;
- **symlink** – родственная жесткой ссылке сущность. Встречаются названия мягкая ссылка и симлинк. В отличие от **hardlink**, – **symlink** это полноценный объект на файловой системе, который, в частности, может переходить ее границы и ссылаться на объекты других ФС. Не увеличивает счетчик ссылок **Links** и не влияет на сохранность адреса ссылки (аналогия с ярлыком Windows) при удалении;
- **pipe** – мы с вами уже встречались с пайпами, **|**, это неименованные пайпы.

```
vagrant@netology1:~$ mkfifo test_pipe
vagrant@netology1:~$ ls -l test_pipe
prw-rw-r-- 1 vagrant vagrant 0 Jul 12 19:03 test_pipe
vagrant@netology1:~$ echo Hello Netology > test_pipe &
vagrant@netology1:~$ cat test_pipe
Hello Netology
```

# Типы файлов, socket

**socket** – бывает сетевым (для соединения удаленных хостов) и локальным (для соединения в рамках одного хоста). Сетевые рассмотрим в следующих лекциях. Локальные (доменные) сокеты – еще один тип межпроцессного взаимодействия (в дополнение к уже известным нам именованным и неименованным пайпам, сигналам и разделяемой памяти процессов). Если в сетевых для подключения сокета используется сетевой адрес+порт, в доменном соquete данным адресом является путь на файловой системе.

## Плюсы перед pipe:

- сокеты могут быть двунаправленными, тогда как pipe – однонаправленные,
- производительнее,
- много клиентов может писать в один pipe, однако сторона-приемник не может узнать, кто записал в pipe, в socket такой проблемы нет.

**Пример:** общение nginx с php-fpm через сокет:

```
root@netology1:~# grep '^listen ' /etc/php/7.4/fpm/pool.d/www.conf
listen = /run/php/php7.4-fpm.sock
```

```
root@netology1:~# grep fastcgi_pass /etc/nginx/sites-enabled/default
fastcgi_pass unix:/run/php/php7.4-fpm.sock;
```

```
root@netology1:~# curl -s 'localhost:81' | grep 'PHP Version' | grep 'td class'
<tr><td class="e">PHP Version </td><td class="v">7.4.3 </td></tr>
```

# Символические ссылки

```
root@netology1:/usr/sbin# ls -l vgs* | head -n2
lrwxrwxrwx 1 root root 3 Feb 13 21:21 vgs -> lvm
lrwxrwxrwx 1 root root 3 Feb 13 21:21 vgscan -> lvm
```

Так ссылки отображаются на ФС. Они могут быть как относительными (в примере выше), так и абсолютными. symlink'и так же создаются командой `ln`, но с ключем `-s`:

```
vagrant@netology1:~$ touch test_file
vagrant@netology1:~$ ln -s $HOME/test_file test_link
vagrant@netology1:~$ ls -l ~/test_link
lrwxrwxrwx 1 vagrant vagrant 23 xxx 18:29 /home/vagrant/test_link ->
/home/vagrant/test_file
```



# Права доступа

# Стандартные права доступа

## Биты разрешений:

- r – read, чтение
- w – write, запись
- x – execution, исполнение

## К кому применяются разрешения:

- владелец
- группа владельца
- все пользователи

Итого, **9 бит** стандартных атрибутов:

Владелец			Группа			Все		
r	w	x	r	w	x	r	w	x
1	1	0	1	0	0	1	0	0

## В десятичной записи:

r:  $2^2 = 4$

w:  $2^1 = 2$

x:  $2^0 = 1$

**644** / rw- r-- r-- / 4 + 2 - 4 - 4

**755** / rwx r-x r-x / 4 + 2 + 1 - 4 + 1 - 4 + 1



---

# Стандартные права доступа, применимость

## Для файлов:

- read для просмотра содержимого файлов
- write для изменения содержимого файлов
- execute для запуска файла в качестве программы (процесса)

## Для директорий:

- read для просмотра содержимого директории (ls)
- write для создания или удаления объектов из директории
- execute для chdir в директорию

# Изменение владельца

chown – change owner, изменить владельца/группу:

```
root@netology1:/tmp# touch file
root@netology1:/tmp# chown vagrant file
root@netology1:/tmp# stat file | grep Uid
Access: (0644/-rw-r--r--) Uid: ( 1000/ vagrant)    Gid: (    0/    root)
```

С одинарным аргументом – меняет пользователя.

С аргументом вида :group – меняет группу.

```
root@netology1:/tmp# chown :vagrant file
root@netology1:/tmp# stat file | grep Uid
Access: (0644/-rw-r--r--) Uid: ( 1000/ vagrant)    Gid: ( 1000/ vagrant)
```

При вызове вида user:, изменит и пользователя и группу (равнозначно user:user):

```
root@netology1:/tmp# touch file2
root@netology1:/tmp# chown vagrant: file2
root@netology1:/tmp# stat file2 | grep Uid
Access: (0644/-rw-r--r--) Uid: ( 1000/ vagrant)    Gid: ( 1000/ vagrant)
```

# Изменение прав доступа

**chmod** – change mode, изменить биты разрешений доступа.

С десятичной записью полных прав: `chmod 0755 file`,  
или с буквенной записью: `chmod u=rwx,g=rx,o=rx file`.

Можно **добавлять (+)** или **убирать (-)** относительно имеющихся:

```
root@netology1:/tmp# stat file | grep Uid
Access: (0755/-rwxr-xr-x)  Uid: ( 1000/  vagrant)    Gid: ( 1000/  vagrant)
root@netology1:/tmp# chmod a-x file
root@netology1:/tmp# stat file | grep Uid
Access: (0644/-rw-r--r--)  Uid: ( 1000/  vagrant)    Gid: ( 1000/  vagrant)
```

## umask – user mask, пользовательская маска

Откуда берутся права доступа по умолчанию?

Для директорий штатные права ОС – 0777, для файлов – 0666. Тем не менее, применяя **mkdir** или **touch**, вы увидите:

```
vagrant@netology1:~$ mkdir dir; touch file
vagrant@netology1:~$ stat {dir,file} | grep Uid
Access: (0775/drwxrwxr-x)  Uid: ( 1000/ vagrant)    Gid: ( 1000/
vagrant)
Access: (0664/-rw-rw-r--)  Uid: ( 1000/ vagrant)    Gid: ( 1000/
vagrant)
```

Ответ: из них вычитается **umask** (0777 - 0002, 0666 - 0002):

```
vagrant@netology1:~$ umask
0002
```

Для привилегированного пользователя значение umask иное – 0022.

# Дополнительные права доступа

Полное поле атрибутов файла – 12 бит.

Перед 9 битами пользовательских разрешений **есть еще три:**

- setuid
- setgid
- sticky

**Логика установки такая же, как с rwx:**

Не-пользовательские атрибуты		
setuid	setgid	sticky
0	0	1

**В десятичной записи:**

setuid:  $2^2 = 4$

setgid:  $2^1 = 2$

sticky:  $2^0 = 1$

```
vagrant@netology1:~$ stat /tmp | grep Uid
Access: (1777/drwxrwxrwt)  Uid: (    0/    root)  Gid: (    0/    root)
```

# Дополнительные права доступа, применимость

**sticky** бит используется для директорий, в которые доступна запись многим пользователям, но требуется защитить файлы от чужого вмешательства.

```
vagrant@netology1:~$ touch /tmp/vagrant_user_file
vagrant@netology1:~$ sudo -u www-data touch /tmp/www_data_user_file
vagrant@netology1:~$ ls -l /tmp/*_user_file
-rw-rw-r-- 1 vagrant vagrant 0 Jul 12 09:15 /tmp/vagrant_user_file
-rw-rw-r-- 1 www-data www-data 0 Jul 12 09:15 /tmp/www_data_user_file
vagrant@netology1:~$ rm /tmp/vagrant_user_file
vagrant@netology1:~$ rm /tmp/www_data_user_file
rm: remove write-protected regular empty file '/tmp/www_data_user_file'?
y
rm: cannot remove '/tmp/www_data_user_file': Operation not permitted
```

**setuid** / **setgid** имеют одинаковый механизм работы за исключением применимости к пользователю (U) или группе (G).

Обычно этот бит можно встретить *на исполняемых файлах*. Пользователь, запускающий такой файл, получает **процесс, работающий с правами владельца исполняемого файла** (а не своими собственными). Классический пример: вызов утилиты смены пароля `passwd`:

```
vagrant@netology1:~$ stat $(which passwd) | grep Uid
Access: (4755/-rwsr-xr-x)  Uid: (   0/   root)  Gid: (   0/   root)
```

# Дополнительные права доступа (продолжение)

passwd требуется обновить файл /etc/shadow, в котором хранятся хеши паролей. Этот файл доступен на запись только root:

```
vagrant@netology1:~$ stat /etc/passwd | grep Uid
Access: (0644/-rw-r--r--)  Uid: (   0/   root)   Gid: (   0/   root)
```

suid бит на исполняемом файле passwd дает возможность запустить его с “эффективными” правами root даже обычному пользователю.

```
vagrant@netology1:~$ ps aux | grep passw[d]
root      242054  0.0  0.3 10464 3344 pts/1    S+   09:50   0:00 passwd
vagrant
vagrant@netology1:~$ grep Uid /proc/242054/status
Uid: 1000 0 0 0 # real, effective
vagrant@netology1:~$ id 0
uid=0(root) gid=0(root) groups=0(root)
vagrant@netology1:~$ id 1000
uid=1000(vagrant) gid=1000(vagrant)
```

Этот же пример показывает опасность некорректного употребления suid/guid, – если бы в passwd не было дополнительных проверок, любой пользователь мог бы поменять пароль root, чего, конечно же, сделать нельзя.

```
vagrant@netology1:~$ passwd root
passwd: You may not view or modify password information for root.
```

# Дополнительные права доступа (продолжение)

**setgid на директориях** имеет другой эффект (setuid обычно не используется). С этим установленным битом вновь создаваемые файлы и директории наследуют группу владельца родительского каталога, а не пользователя, инициировавшего операцию.

```
vagrant@netology1:~$ mkdir test_dir
vagrant@netology1:~$ sudo chown :tcpdump test_dir
vagrant@netology1:~$ stat test_dir/ | grep Gid
Access: (0775/drwxrwxr-x)  Uid: ( 1000/ vagrant)    Gid: (65534/ nogroup)
```

## Стандартное поведение без setgid:

```
vagrant@netology1:~$ mkdir test_dir/no_gid; stat $_ | grep Gid
Access: (0775/drwxrwxr-x)  Uid: ( 1000/ vagrant)    Gid: ( 1000/ vagrant)
```

## Поведение с установленным setgid:

```
vagrant@netology1:~$ chmod g+s test_dir/
vagrant@netology1:~$ stat test_dir/ | grep Uid
Access: (2775/drwxrwsr-x)  Uid: ( 1000/ vagrant)    Gid: ( 114/ tcpdump)
vagrant@netology1:~$ mkdir test_dir/with_gid; stat $_ | grep Gid
Access: (2775/drwxrwsr-x)  Uid: ( 1000/ vagrant)    Gid: ( 114/ tcpdump)
```



# Дополнительные атрибуты, lsattr/chattr

Помимо стандартных 12 бит для записи разрешений в метаданных файла, могут присутствовать дополнительные атрибуты, способные ввести в затруднение тех, кто о них не знает:


```
root@netology1:~# echo 'test' > test_file
-bash: test_file: Operation not permitted
root@netology1:~# stat test_file | grep Uid
Access: (0644/-rw-r--r--)  Uid: (   0/   root)   Gid: (   0/   root)
```

Почему root не может записать данные в файл, которым он владеет и имеет права на запись?

```
root@netology1:~# lsattr test_file
----i-----e----- test_file
```

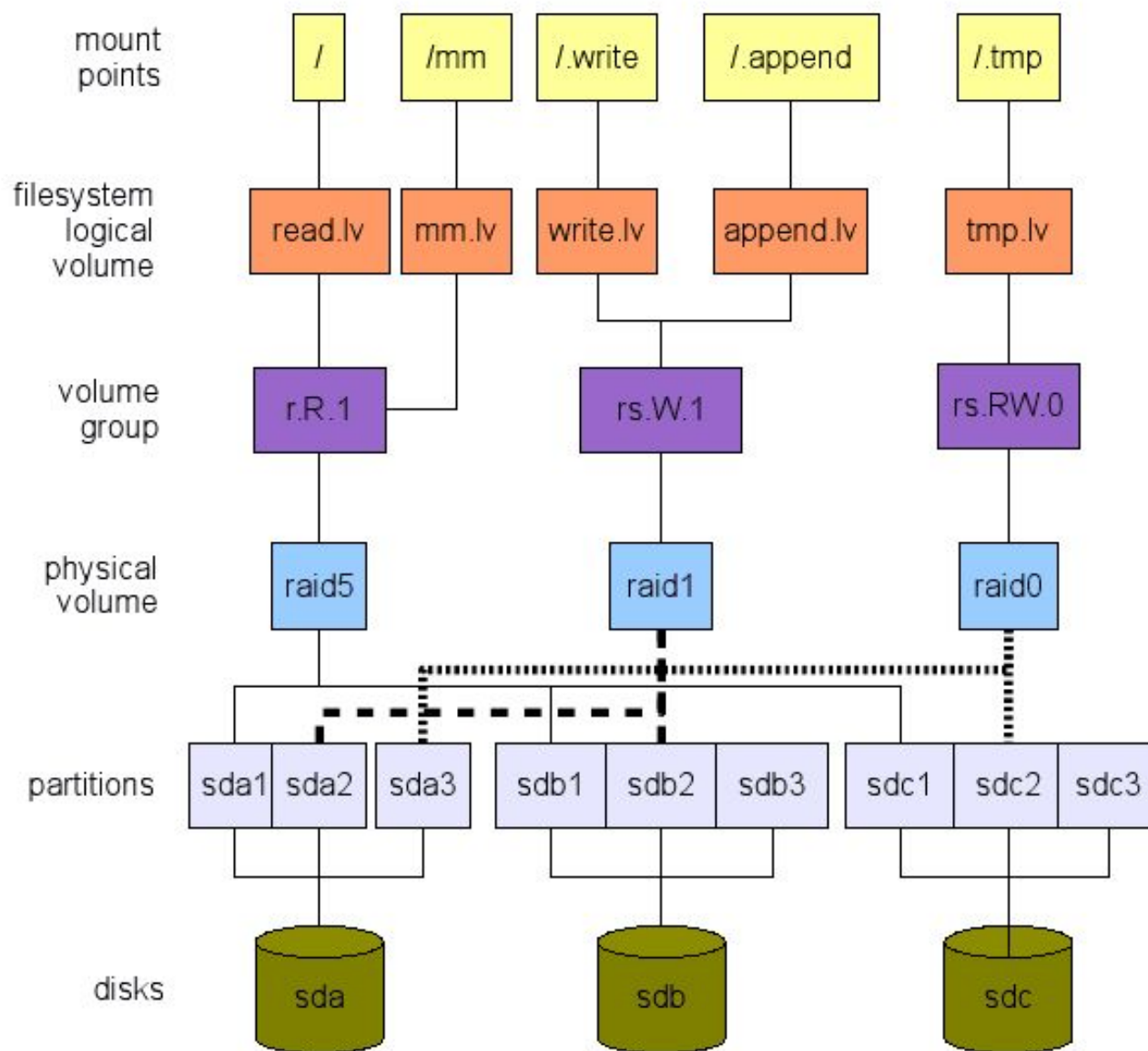
Установлен флаг **i** – immutable, неизменяемый. Полный список расширенных атрибутов – `man chattr`, `chattr -i` для удаления immutable:

```
root@netology1:~# chattr -i test_file
root@netology1:~# echo 'test' > test_file
root@netology1:~# echo $?
0
```



# Организация хранения данных в Linux

# Схема уровней хранения данных



# Базовые уровни хранения данных

- Блочное устройство (одиночный диск, RAID массив) – **/dev/sda**, **/dev/nvme0n1**:

```
:~# lsblk
NAME                MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
nvme2n1              259:2    0   8G  0 disk
```

- Таблица разделов, раздел на блочном устройстве – **/dev/sda1**, **/dev/nvme2n1p1**:

```
nvme2n1              259:2    0   8G  0 disk
└─nvme2n1p1          259:3    0   8G  0 part /
```

- Файловая система на разделе:

```
/dev/nvme2n1p1 on / type ext4 (rw,relatime,discard)
```

Большая гибкость ОС позволяет добавлять или заменять уровни: например, вместо создания таблицы разделов непосредственно на физическом блочном устройстве, можно использовать менеджер логических томов LVM2, и размещать файловую систему на его томах.

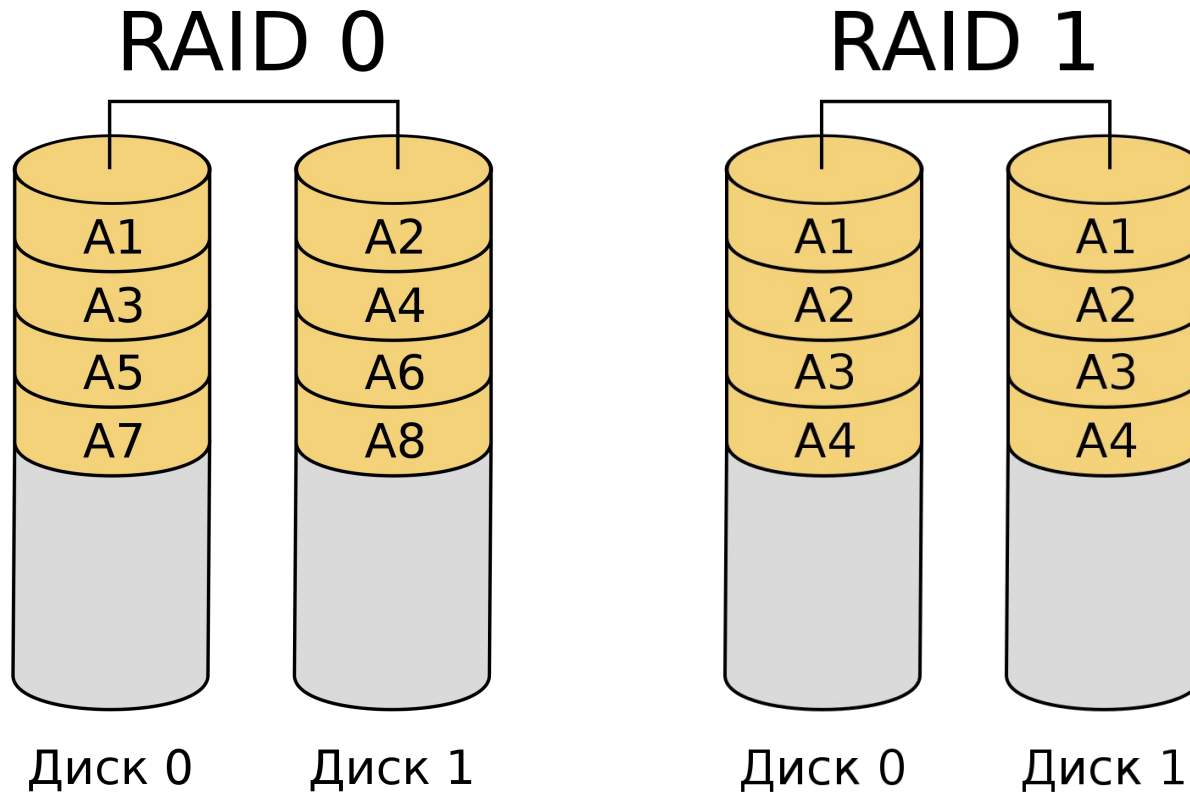
Или же, таблицу разделов можно не использовать вовсе, разместив ФС напрямую на блочном устройстве.



# RAID/mdadm

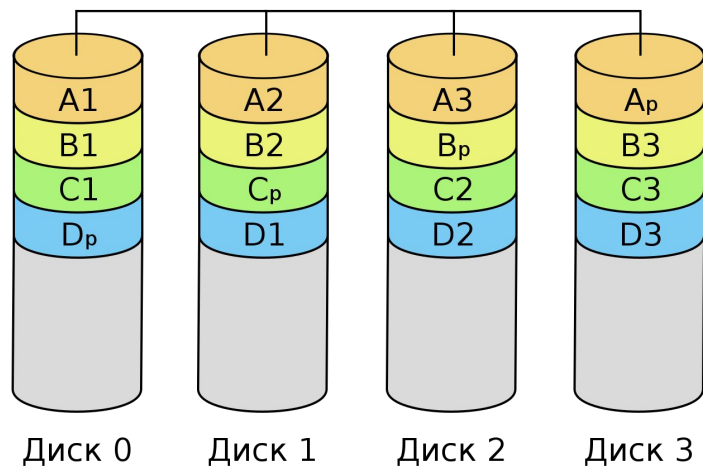
# RAID0, RAID1

**RAID** – **R**edundant **A**rray of **I**ndependent **D**isks,  
избыточный массив независимых дисков.



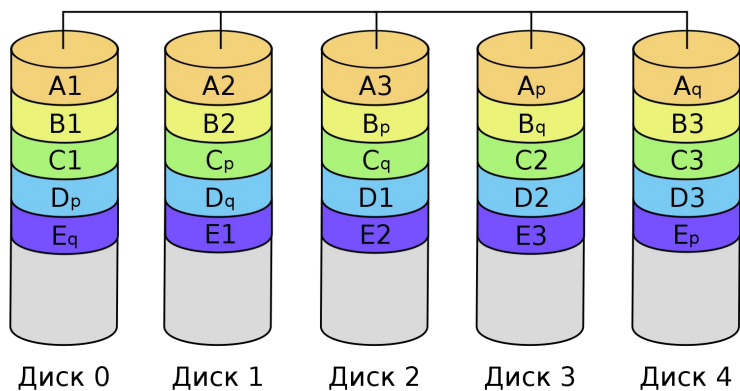
# RAID5, RAID6

## RAID 5



A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

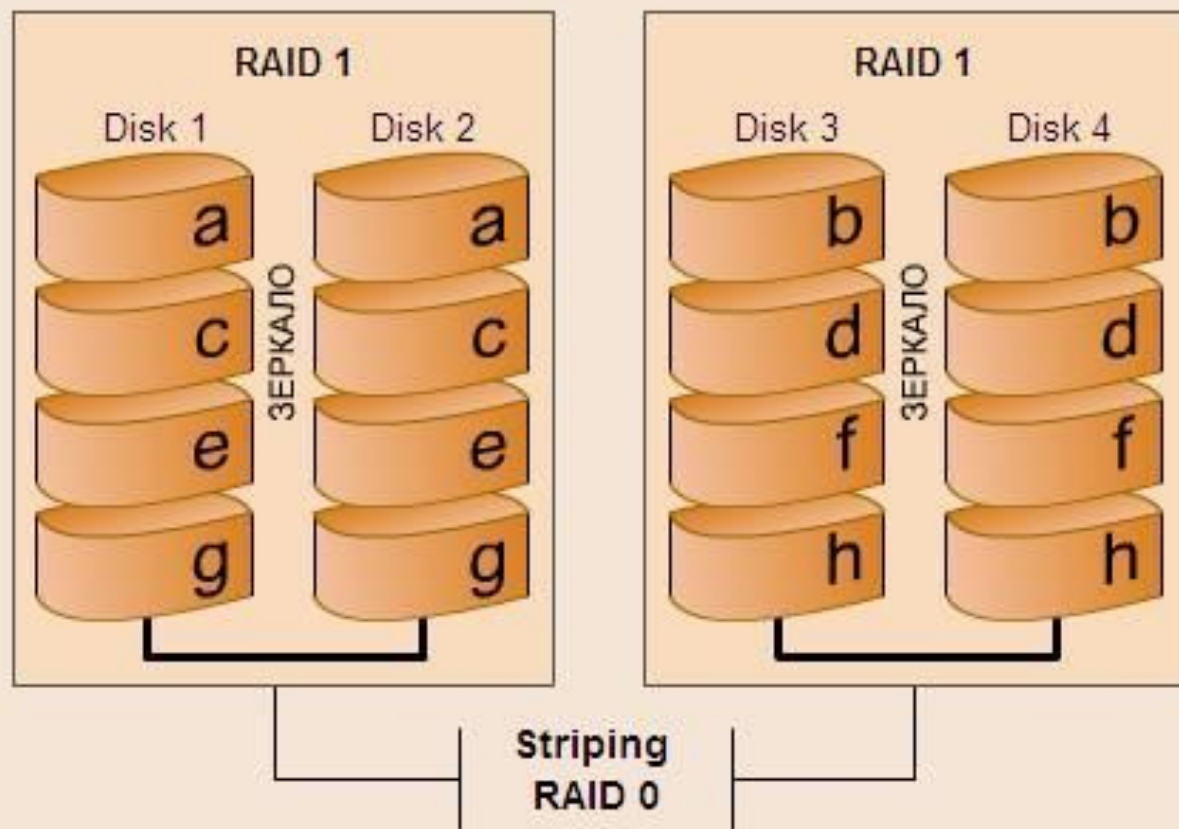
## RAID 6



# RAID10 (1+0)

## RAID 10

Отказоустойчивый массив с дублированием и параллельной обработкой





# mdadm на примере RAID0 Amazon EBS

EC2 инстанс с 2 SSD дисками по 5 Гб каждый:

```
nvme0n1      259:0      0    5G    0 disk
nvme1n1      259:1      0    5G    0 disk
```

Создаем файловую систему на одиночном диске и тестируем производительность случайной записи:

```
mkfs.ext4 /dev/nvme1n1
mkdir /nvme_mount
mount /dev/nvme1n1 /nvme_mount
cd /nvme_mount; fio --name=randwrite --ioengine=libaio --iodepth=1 --rw=randwrite
--bs=4k --direct=0 --size=512M --numjobs=2 --runtime=60 --group_reporting
```

**Результат:**

```
WRITE: bw=9873KiB/s (10.1MB/s), 9873KiB/s-9873KiB/s (10.1MB/s-10.1MB/s), io=427MiB
(448MB), run=44280-44280msec
```

Если вы посмотрите документацию, обнаружите, что без существенного увеличения размера диска (а, значит, и стоимости), получить больше производительности не получится.

**На помощь придет mdadm:**

```
mdadm --create --verbose /dev/md0 --level=0 --raid-devices=2 /dev/nvme1n1 /dev/nvme2n1

WRITE: bw=19.9MiB/s (20.8MB/s), 19.9MiB/s-19.9MiB/s (20.8MB/s-20.8MB/s), io=1024MiB
(1074MB), run=51507-51507msec
```

# Статус mdadm

```
:~# cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5]
[raid4] [raid10]
md0 : active raid1 nvme1n1[1] nvme0n1[0]
      5237760 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

Искусственно пометим один из дисков сбойным:

```
:~# mdadm --fail /dev/md0 /dev/nvme1n1
mdadm: set /dev/nvme1n1 faulty in /dev/md0
root@ip-10-10-5-245:~# cat /proc/mdstat
Personalities : [linear] [multipath] [raid0] [raid1] [raid6] [raid5]
[raid4] [raid10]
md0 : active raid1 nvme1n1[1](F) nvme0n1[0]
      5237760 blocks super 1.2 [2/1] [U_]

unused devices: <none>
```

# mdadm.conf

```
root@vagrant:~# egrep '(DEVICE|initram)' /etc/mdadm/mdadm.conf
# !NB! Run update-initramfs -u after updating this file.
# !NB! This will ensure that initramfs has an uptodate copy.
#DEVICE partitions containers
```

Воспользуемся инструкцией для сохранения конфигурации:

```
root@vagrant:~# echo 'DEVICE partitions containers' > /etc/mdadm/mdadm.conf
root@vagrant:~# mdadm --detail --scan >> /etc/mdadm/mdadm.conf

root@vagrant:~# cat /etc/mdadm/mdadm.conf
DEVICE partitions containers
DEVICE partitions containers
ARRAY /dev/md0 metadata=1.2 name=vagrant:0
UUID=0e10b2a4:4b4a20e3:f3b7cc8d:a16a3425

root@vagrant:~# update-initramfs -u
update-initramfs: Generating /boot/initrd.img
```

Проверим после перезагрузки:

```
root@vagrant:~# mkdir /mountpoint
root@vagrant:~# blkid | grep md0
/dev/md0: UUID="9555ea98-78f3-4726-86c1-f9bd25916a12" TYPE="ext4"
root@vagrant:~# echo 'UUID=9555ea98-78f3-4726-86c1-f9bd25916a12 /mountpoint
ext4 errors=remount-ro 0 1' >> /etc/fstab; reboot
```



# LVM2

# Концепции менеджера логических томов

- физические тома – **Physical Volumes**, **pvd**isplay/pvcreate/pvmove/...  
(обычно размещаются на блочных устройствах)
- Физические тома объединяются в группы томов – **Volume Groups**, **vg**display/vgcreate/...
- VG служат пулом для логических томов – **Logical Volumes**, **lv**display/lvcreate/lvremove/...

```
root@netology1:~# pvs
PV          VG          Fmt  Attr  PSize   PFree
/dev/sda5   vgvagrant lvm2  a--   <63.50g    0
root@netology1:~# vgs
VG          #PV #LV #SN Attr   VSize   VFree
vgvagrant   1   2   0 wz--n- <63.50g    0
root@netology1:~# lvdisplay /dev/vgvagrant/root
--- Logical volume ---
LV Path                /dev/vgvagrant/root
LV Name                 root
VG Name                 vgvagrant
LV UUID                 0v7tWH-gBt5-oNrf-MTrG-iCme-ipGm-BAf0EU
...
```



# Таблица разделов

# Просмотр и копирование таблицы разделов

```
:~# fdisk -l /dev/nvme2n1
Disk /dev/nvme2n1: 8 GiB, 8589934592 bytes, 16777216 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0xb93804ca
```

Device	Boot	Start	End	Sectors	Size	Id	Type
/dev/nvme2n1p1	*	2048	16777182	16775135	8G	83	Linux


**sfdisk** также работает в интерактивном режиме для модификации таблицы разделов.

Для копирования таблицы разделов на другое устройство лучше воспользоваться **sfdisk**:

```
:~# sfdisk -d /dev/nvme2n1
label: dos
label-id: 0xb93804ca
device: /dev/nvme2n1
unit: sectors

/dev/nvme2n1p1 : start=          2048, size=      16775135, type=83, bootable

:~# sfdisk -d /dev/nvme2n1 | sfdisk <new_disk>
```



# **Создание и монтирование файловых систем**



## man 5 fstab

Сама по себе созданная на разделе, томе или устройстве файловая система не появится в ОС автоматически – ее необходимо смонтировать (примерно как назначит букву диска в Windows.)

```
vagrant@netology1:~$ grep -v ^# /etc/fstab | column -t
/dev/mapper/vgvagrant-root      /                ext4  errors=remount-ro
0  1
UUID=E4AD-0D53                  /boot/efi        vfat  umask=0077
0  1
/dev/mapper/vgvagrant-swap_1    none             swap  sw
0  0
```

Для создания файловых систем: `mkfs.<fs_name>`, `mkfs.ext4`.

---

## Итоги

- узнали о структурах файловой системы и типах устройств;
- поговорили об организации хранения данных в Linux;
- рассмотрели наиболее распространенные уровни RAID;
- посмотрели на практике реализацию mdadm;
- поработали с менеджером логических томов LVM2.



## Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера .
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

**Задавайте вопросы и  
пишите отзыв о лекции!**

**Сергей Андрюнин**