Шпаргалка по командам Linux

34 карточек: короткий пример кода и подпись, где обычно ошибаются. Свежая версия, фильтр и песочница — по QR.

koddo.ru/cheatsheets/linux

Файлы и навигация в Linux

Навигация, создание, копирование и удаление — команды первого дня.

Навигация: pwd, ls, cd

```
pwd              # /work
ls               # projects  reports
ls -a            # + скрытые: .  ..  .git
cd projects/blog # относительный путь
cd /work         # абсолютный, от корня
cd -             # назад, где был
```

Относительный путь считается от текущего каталога, абсолютный начинается с /. Скрытые файлы — любые имена с точки: ls их прячет без -a.

Создание: mkdir -p и touch

```
mkdir docs                 # один каталог
mkdir -p reports/2026/aug  # вся цепочка сразу
touch notes.txt            # пустой файл
```

Без -p mkdir падает, если родительских каталогов нет, и ругается File exists на повторный запуск. touch у существующего файла лишь обновляет время изменения.

Копирование и переименование: cp, mv

```
cp draft.md final.md   # копия файла
cp -r projects backup  # каталог — только с -r
mv final.md docs/      # переместить
mv draft.md intro.md   # переименовать (та же команда)
```

Переименование и перемещение — одна операция mv: меняется путь. cp без -r молча пропускает каталоги: «-r not specified; omitting directory».

Удаление: rm и rm -r

```
rm notes.txt      # файл
rm backup         # каталог так нельзя: Is a directory
rm -r backup      # каталог со всем содержимым
ls build/*.tmp    # СНАЧАЛА проверить маску
rm build/*.tmp    # потом удалять
```

Корзины в терминале нет: rm -r удаляет сразу и навсегда, а rm -rf не переспросит даже про защищённые файлы. Перед удалением по маске прогони её через ls.

Просмотр: cat, less, head, tail

```
cat notes.txt      # файл целиком
less app.log       # постранично: / поиск, q выход
head -n 20 app.log # первые 20 строк
tail -n 20 app.log # последние 20
tail -f app.log    # следить за живым логом
```

cat хорош для коротких файлов, длинные листай в less. tail -f не завершается и печатает дописываемые строки — стандартный способ смотреть логи; выход Ctrl+C.

Справка: man, --help, which

```
man ls       # полное руководство, q выход
ls --help    # короткая выжимка флагов
which ls     # /usr/bin/ls — что запускается
```

man листается как less: /ищет, q выходит. which — первый шаг диагностики «команда ведёт себя странно»: показывает, какой именно файл выполняется вместо ожидаемого.

Поиск в Linux: find и grep

find отвечает «какие файлы», grep — «какие строки внутри».

find: поиск файла по имени

```
find . -name "*.log"    # по маске от текущего каталога
find . -iname "APP.LOG" # без учёта регистра
find / -name "config.yaml" 2>/dev/null  # по всей системе
```

Маску всегда бери в кавычки: без них шелл раскроет звёздочку по текущему каталогу до запуска find. Хвост 2>/dev/null прячет Permission denied при поиске от корня.

find: по типу и свойствам

```
find . -type d            # только каталоги
find . -type f -name "*.py"  # только файлы
find . -type f -mtime -1  # изменённые за сутки
find . -type f -size +100M   # больше 100 МБ
```

Условия комбинируются простым перечислением — это логическое И. -mtime считает сутками: -1 значит «моложе 24 часов», +7 — «старше недели».

find -exec: обработать найденное

```
find . -name "*.log" -exec wc -l {} +  # строки в каждом
find . -name "*.tmp"                   # 1) проверить глазами
find . -name "*.tmp" -exec rm {} +     # 2) удалить
```

{} — место подстановки найденного, + собирает все находки в один вызов команды. Перед разрушающим -exec запусти тот же find без него и проверь список.

grep: поиск текста в файле

```
grep "ERROR" app.log     # строки с ERROR
grep -n "ERROR" app.log  # + номера строк
grep -in "error" app.log # без учёта регистра
```

grep по умолчанию чувствителен к регистру: без -i строки с error в нижнем регистре молча выпадут из выдачи, и пропажу легко не заметить.

grep -r: поиск по проекту

```
grep -rn "TODO" .        # рекурсивно, файл:строка
grep -rl "TODO" src/     # только имена файлов
grep -rn --exclude-dir=node_modules --exclude-dir=.git "TODO" .
```

Каталог grep сам не обходит — без -r ответит Is a directory. В репозиториях сразу исключай .git и node_modules, иначе выдачу зальёт служебными совпадениями.

grep -v и -c: фильтр и счёт

```
grep -v "INFO" app.log   # строки БЕЗ образца
grep -c "INFO" app.log   # только число совпавших строк
ps aux | grep -v grep | grep nginx  # классика фильтров
```

-v инвертирует условие, -c печатает счётчик вместо строк. В связке ps | grep сам grep попадает в выдачу — его выкидывают через -v grep (или берут pgrep).

Конвейеры и потоки в Linux

Перенаправление вывода, поток ошибок и связки команд через |.

Перенаправление: > и >>

```
echo "молоко" > list.txt   # перезаписать файл
echo "хлеб" >> list.txt    # дописать в конец
sort data.txt > data.txt   # ЛОВУШКА: файл очистится
sort -o data.txt data.txt  # сортировка на месте — так
```

Файл после > очищается до запуска команды, поэтому перенаправление в тот же файл уничтожает данные: sort уже читает пустоту. Для записи «на место» у sort есть -o.

Потоки ошибок: 2>&1 и /dev/null

```
cmd > out.log        # только stdout, ошибки на экране
cmd > all.log 2>&1   # оба потока в файл
cmd 2>/dev/null      # спрятать ошибки
cmd 2>&1 > out.log   # ЛОВУШКА: ошибки мимо файла
```

У процесса два выходных потока: stdout (1) и stderr (2). Порядок важен: 2>&1 значит «туда, куда 1 смотрит сейчас», поэтому пишется после > файла, не до.

Конвейер: sort | uniq -c

```
sort tags.txt | uniq -c | sort -nr
#   3 python
#   2 sql
#   1 linux  — частотный словарь тремя командами
```

| соединяет stdout левой команды со stdin правой. uniq схлопывает только соседние повторы, поэтому перед ним обязателен sort — иначе дубли молча останутся.

Подсчёт: wc -l

```
wc -l app.log            # строк в файле
ls | wc -l               # записей в каталоге
grep -r "TODO" . | wc -l # пометок в проекте
```

wc -l считает строки, -w слова, -c байты. В конвейере это универсальный финальный шаг «сколько всего» для вывода любой команды.

tee: на экран и в файл разом

```
./deploy.sh | tee deploy.log     # видно и записано
./deploy.sh | tee -a deploy.log  # дописывать, не перезаписывать
```

tee дублирует поток: печатает на экран и одновременно пишет в файл. Без -a каждый запуск перезапишет лог предыдущего — для накопления флаг обязателен.

Цепочки: && и ||

```
mkdir -p build && cd build   # cd только при успехе
npm test || echo "тесты упали"  # правая часть при провале
cmd1 ; cmd2                  # просто подряд, без условий
```

&& запускает правую команду только при успехе левой (код возврата 0), || — только при провале. Точка с запятой условий не ставит: выполнится всё подряд.

Права доступа в Linux

Триады rwx, числовая запись chmod и владельцы файлов.

Читаем ls -l: rwx по ролям

```
ls -l deploy.sh
# -rwxr-xr-x 1 alice dev 41 Aug 12 10:00 deploy.sh
#  ^владелец ^группа ^остальные   (r читать w писать x выполнять)
```

Первый символ — тип (- файл, d каталог, l ссылка), дальше три триады: владелец, группа, остальные. На каталоге x значит «право войти», а r — лишь читать список имён.

chmod числами: 755, 644, 600

```
chmod 755 deploy.sh  # rwxr-xr-x — скрипты, каталоги
chmod 644 notes.txt  # rw-r--r-- — обычные файлы
chmod 600 id_rsa     # rw------- — приватные ключи
```

Каждая цифра — сумма прав одной роли: r=4, w=2, x=1; семёрка — всё, пятёрка — чтение и выполнение. chmod 777 «чинит» Permission denied ценой записи для любого процесса — не решение.

chmod буквами: u+x, g-w

```
chmod +x script.sh   # исполняемый — самый частый вызов
chmod u+x tool.sh    # x только владельцу
chmod g-w shared.txt # забрать запись у группы
./script.sh          # запуск: ./ обязателен
```

u владелец, g группа, o остальные, a все; + добавить, - забрать. Точка со слэшем при запуске обязательна: без ./ шелл ищет команду в PATH, а не в текущем каталоге.

chown: владелец и группа

```
sudo chown alice report.txt      # сменить владельца
sudo chown alice:dev report.txt  # владельца и группу
sudo chown -R alice:dev /srv/app # рекурсивно
```

chmod меняет биты, chown — кому они принадлежат; менять владельца может только root. Связка «группа dev + права 640» — стандартный способ дать доступ команде, а не людям поодиночке.

umask: права по умолчанию

```
umask        # 0002 — маска Ubuntu для пользователя
touch a.txt  # получит 664 (666 - маска)
umask 027    # строже: новые файлы 640
touch b.txt  # rw-r-----
```

Новые файлы стартуют от 666, каталоги от 777, маска вычитает биты. На серверах обычна маска 0022 (файлы 644); umask 027 закрывает файлы от посторонних с момента создания.

Процессы в Linux

Посмотреть, запустить в фоне, остановить — и прочитать код возврата.

Посмотреть процессы: ps, pgrep

```
ps aux                   # все процессы системы
ps -o pid,stat,cmd -p 42  # конкретный PID
pgrep -a nginx           # PID + команда по имени
```

ps aux — снимок всей системы: USER, PID, %CPU, %MEM, STAT, COMMAND. pgrep ищет по имени и, в отличие от ps | grep, не ловит в выдачу сам себя.

Фон: &, $! и wait

```
sleep 300 &   # запустить в фоне
echo $!       # PID последнего фонового
wait $!       # дождаться его завершения
```

& возвращает управление сразу, процесс работает в фоне. $! — PID последнего фонового процесса: сохрани его сразу, следующий & перезапишет значение.

kill: сначала TERM, потом -9

```
kill 4242     # SIGTERM: вежливо, можно убраться
kill -9 4242  # SIGKILL: ядро снимает без уборки
pgrep -a node # перед pkill — проверь, кого убьёшь
```

kill без номера шлёт SIGTERM — процесс может дописать данные и выйти. SIGKILL не перехватывается: недописанные файлы и незакрытые транзакции останутся. Поэтому -9 — последний аргумент.

Сигналы и trap

```
# TERM(15) стоп · KILL(9) насильно · INT(2) Ctrl+C · HUP(1)
trap 'rm -f "$tmp"; exit' TERM INT  # уборка при остановке
kill -l   # весь список сигналов
```

SIGTERM и SIGINT перехватываемы — на этом строится graceful shutdown (docker stop и systemctl stop шлют именно TERM). SIGKILL и SIGSTOP перехватить нельзя.

Коды возврата: $? и 128+N

```
ls /nope; echo $?  # 2 — ошибка
# убит сигналом → 128 + номер:
# 143 = TERM(15) · 137 = KILL(9, часто OOM) · 130 = Ctrl+C(2)
```

0 — успех, всё остальное — ошибка; $? хранит код последней команды. Числа выше 128 читаются как «снят сигналом»: 137 в логах CI и Docker — почти всегда нехватка памяти.

Система: top, df -h, du -sh

```
top        # живая таблица процессов, q выход
df -h      # свободное место по дискам
du -sh logs/  # сколько весит каталог
```

top — динамика вместо снимка ps; удобнее его htop, если установлен. df смотрит на файловые системы целиком, du — на конкретный каталог: пара для вопроса «куда делось место».

Пользователи и sudo в Linux

Учётные записи, группы и повышение привилегий без постоянного root.

Кто я: whoami, id, groups

```
whoami   # alice
id       # uid=1001(alice) gid=1001(alice) groups=...,1002(dev)
groups   # alice dev
```

uid — числовой идентификатор пользователя, gid — его первичная группа; root — это uid 0, для него проверки прав не выполняются. id alice покажет то же про другого.

sudo: одна команда от root

```
sudo systemctl restart nginx  # спросит ТВОЙ пароль
sudo -i                       # целая root-сессия (реже)
```

sudo выполняет команду от root по твоему паролю и пишет каждое использование в журнал — поэтому он безопаснее постоянной работы под root, где любая опечатка в rm необратима.

Создать пользователя: useradd -m

```
sudo useradd -m -s /bin/bash alice  # с домом и bash
sudo passwd alice                   # задать пароль
grep "^alice" /etc/passwd           # проверить запись
```

Без -m домашний каталог не создастся, без -s шеллом будет /bin/sh. Учётки лежат строками в /etc/passwd, хэши паролей — в закрытом /etc/shadow.

Группы: usermod -aG (только с -a!)

```
sudo groupadd dev
sudo usermod -aG dev alice  # ДОБАВИТЬ в группу
# usermod -G dev alice — ЗАМЕНИТ весь список групп!
id alice                    # проверить членство
```

Флаг -a означает append: без него -G заменяет список дополнительных групп целиком, и пользователь молча вылетает из sudo и docker. Новые группы процесс получает только после перелогина.

su: сессия под другим пользователем

```
su - alice   # войти как alice (пароль alice)
whoami       # alice
exit         # вернуться в свою сессию
```

Дефис у su - важен: он даёт полноценное окружение пользователя, а не только имя. Для админ-задач предпочитай sudo — он точечный и оставляет след в журнале.