

Инструменты разработчика. Вопросы.

0. Основные команды оболочки и работа в терминале Linux

(a) Основные команды

- `pwd`
- `cd`
- `ls` и её опции `-l`, `-a` и `-i`
- `echo` и её опции `-n` и `-e`
- `cat` и её опция `-n`.
- `less` и её опции `-N` и `-R`. Горячие клавиши Пробел, `b`, `q`, `g`, `G`, `n`, `N`, `h`. Поиск текста в файле
- Текстовый редактор `nano`
- `file`
- `alias`, создание и удаление алиасов. Просмотр всех алиасов. Как сохранить алиас навсегда?
- `man`, поиск в конкретном разделе
- `date`, печать даты/времени в форматированном виде (опция `+%формат%`)

(b) Команды для манипуляции с файлами

- `touch`
- `mkdir` и её опция `-p`
- `cp` и её опции `-r`, `-a`, `-t`, `-i`, `-u`, `-n`
- `mv`, переименование файлов с помощью `mv`
- `rm` и её опции `-r`, `-f`

(c) Сокращение имён директорий

- `.`
- `..`
- `~`
- `~alice`

(d) Горячие клавиши, используемые в терминале

- `Ctrl-C`
- `Ctrl-D`
- `Ctrl-Z`

(e) Команды для архивации

- `gzip` и её опции `-d` и `-k`
- `bzip2` и её опции `-d` и `-k`
- `tar` и её опции `-c`, `-x`, `-t`, `-v`, `-f`, `-z` и `-j`

(f) Пакетные менеджеры

Пакетные менеджеры `apt` и `dnf`. Подкоманды этих пакетных менеджеров:

- `install/remove`
- `update/upgrade`
- `search`

1. Основы git

(a) Системы контроля версий

Что такое система контроля версий? Локальные, централизованные и распределённые системы контроля версий. Отличие `git` от других систем контроля версий.

(b) Настройка git

Команда `git config`. Печать всех настроек `git`. Добавление и удаление настроек. Уровни настроек:

- Системный
- Глобальный
- Локальный

Файлы конфигурации, в которых хранятся эти настройки. Основные настройки:

- `user.name`
- `user.email`
- `core.editor`
- `core.autocrlf`
- `alias.<name>`

(c) **Создание репозитория**

Команда `git init` и её опция `--bare`. Команда `git clone`.

(d) **Работа с файлами и коммитами**

Рабочая директория. Индекс. Команда `git add`. Добавление всех изменений из рабочей директории в индекс. Удаление файлов из индекса. Команда `git rm` и её опция `--cached`. Локальный репозиторий. Коммит. Хэш коммита. Команда `git commit` и её опция `-m`.

(e) **Команды для просмотра информации индекса и локального репозитория**

- `git status`
- `git show`
- `git diff` и её опция `--staged`
- `git log` и её опции `--oneline`, `--graph`, `--all`, `--pretty=format`
- `git log` для поиска в истории, её опции `--since`, `--author`, `--grep`, `-S`
- `git blame`

(f) **Работа с ветками**

Что такое ветка? Основная ветка `main`. Команда `git branch` и её опции `-d`, `-D`, `-m`, `-M`, `-v`, `-r`, `-vv`.

(g) **Адресация коммитов**

Адресация коммитов с использованием хэша. Полный и сокращённый хэш коммита. Адресация коммитов с помощью веток и указателя `HEAD`. Символы `~` и `^`.

(h) **Перемещение по графу коммитов**

Указатель `HEAD`. Команда `git switch` и её опции `-c` и `--detach`. Чем `git switch` отличается от старой команды `git checkout`? Переход на отдельный коммит. Состояние `detached HEAD` и чем опасно это состояние?

(i) **Слияние**

Слияние веток. Команда `git merge`. Слияние перемоткой (`fast-forward merge`) и когда используется такой вид слияния. Опция `--no-ff`. Конфликты при слиянии. Когда возникают конфликты? Как разрешить конфликт? Команды

- `git merge --abort`
- `git merge --continue`

Как отменить произведённое слияние?

(j) **Перебазирование**

Перебазирование веток. Команда `git rebase`. Отличие перебазирования от слияния. Преимущества и недостатки перебазирования по сравнению со слиянием. Конфликты при перебазировании. Как разрешить конфликт, возникший при перебазировании? Команды:

- `git rebase --abort`
- `git rebase --continue`
- `git rebase --skip`

Как отменить произведённое перебазирование?

2. Продолжение git

a. **Откат состояния**

Команда `git reset` и её режимы `--soft`, `--mixed` и `--hard`. Как меняется рабочая директория, индекс и локальный репозиторий при использовании `git reset` в каждом из этих режимов? Как отменить произведённый откат состояния? Команда `git reflog`, что она показывает? В каких случаях использование `git reset` может привести к безвозвратной потере данных? Команда `git restore` и её опция `--source`. Как восстановить случайно удалённый файл в рабочей директории?

b. Копирование отдельных коммитов

Команда `git cherry-pick`. В чём недостатки использования `git cherry-pick`? Конфликты при копировании коммитов. Команды:

- `git cherry-pick --abort`
- `git cherry-pick --continue`
- `git cherry-pick --skip`

c. Интерактивное перебазирование

Команда `git rebase -i`. Файл `git-rebase-todo`. Команды интерактивного перебазирования:

- | | |
|-----------------------|-----------------------|
| • <code>pick</code> | • <code>squash</code> |
| • <code>reword</code> | • <code>fixup</code> |
| • <code>edit</code> | • <code>drop</code> |

Конфликты при интерактивном перебазировании. Как отменить произведённое интерактивное перебазирование?

d. Типы файлов в git

- Отслеживаемые (tracked)
 - Индексированные (staged)
 - Неизменённые (unmodified)
 - Изменённые (modified)
- Неотслеживаемые (untracked)
- Игнорируемые (ignored)

e. Игнорируемые файлы

Файл `.gitignore` и как с его помощью:

- Игнорировать все файлы в репозитории с данным именем
- Игнорировать все директории в репозитории с данным именем
- Игнорировать один конкретный файл
- Игнорировать файлы по шаблону `*`, `?`, `[...]`
- Сделать исключение из игнорирования

Игнорирование пустых директорий. Как заставить `git` не игнорировать пустые директории?

f. Очистка репозитория

На какие типы файлов не действует команда `git reset --hard`? Какие типы файлов меняются при использовании `git switch`? Команда `git clean` и её опции `-f`, `-d`, `-x`, `-X`, `-n`.

g. Удалённый репозиторий

Удалённый репозиторий. Ремоут (remote). В чём разница между удалённым репозиторием и ремоутом? Команда `git remote`, её опция `-v` и подкоманды `add`, `remove`, `rename`, `set-url` и `show`.

h. Удалённые ветки

Что такое удалённая ветка (remote branch)? Что такое ветка отслеживания (remote-tracking branch)? Какие имена имеют ветки отслеживания в `git`? На что указывают такие ветки? Когда обновляется состояние таких веток? Команда:

- `git branch -a`

i. Работа с удалённым репозиторием

Команды для взаимодействия локального репозитория с удалённым:

- `git push` и её опции `--all`, `--tags`, `-f`, `--force-with-lease`, `-u`, `--set-upstream`.
- `git fetch` и её опции `--prune` и `-tags`.
- `git pull` и её опции `--rebase` и `--ff-only`. Конфликты при использовании `git pull`.

Чем `git pull` отличается от `git fetch`? Перезапись истории в удалённом репозитории. Отмена изменений в удалённом репозитории. Команда `git push -f` и в чём её опасность? Команда `git revert`.

j. Отслеживающие и upstream-ветки

Что такое отслеживающая ветка (tracking branch)? Что такое upstream-ветка (upstream-branch)? Как привязать отслеживающую ветку к upstream-ветке? Какие преимущества даёт такая привязка? Что будет, если не сделать такую привязку? Команды:

- `git branch -vv`
- `git branch --set-upstream-to=origin/main`
- `git branch --unset-upstream`
- `git push -u origin main`

k. Тэги

Чем тэги отличаются от веток? Зачем нужны тэги? Команда `git tag` и её опции `-d`, `-l`, `-a` и `-m`.

l. Pull request

Веб-сервисы для хранения и работы с удалёнными репозиториями. GitHub. GitLab. Форк. Pull request. Merge request.

3. Продвинутый git (не будет на коллоквиуме)

4. Потоки и конвейеры

a. Основные команды, которые часто используются в конвейерах

- `tac`
- `head` и её опции `-n` и `-c`
- `tail` и её опции `-n`, `-c`, `-f` и `-F`
- `uniq` и её опция `-c`
- `sort` и её опции `-n`, `-r`, `-u`, `-h`, `-k` и `-t`.
- `wc` и её опции `-l`, `-w`, `-c` и `-m`

b. Потоки и перенаправление

Стандартные потоки ввода вывода:

- `stdin`
- `stdout`
- `stderr`

Куда эти потоки направлены по умолчанию? Перенаправление потока в файл для перезаписи. Перенаправление потока в файл для дозаписи. Перенаправление файла в `stdin`. Перенаправление `stderr` в файл. Как перенаправить `stdout` и `stderr` в один файл? Файл `/dev/null`. Что произойдёт, если перенаправить потоки `stdout/stderr` в этот файл? Что произойдёт если перенаправить файл `/dev/null` в `stdin`?

c. Расширения оболочки

- Шаблон поиска (wildcard/glob pattern). Wildcard-символы:
 - `*`
 - `?`
 - `[...]`
 - `[!...]`
- Brace expansion `{...}`.
- Синтаксис подстановки команд `$(...)`.
- Подстановка процесса `<(...)` и `>(...)`. Примеры использования подстановки процесса.

d. Конвейеры

Что такое конвейер оболочки Bash и как его использовать? Примеры простых конвейеров. Код возврата конвейера. Переменная `PIPESTATUS`. Как перенаправить поток `stderr` в конвейер?

e. Команда find

Опции команды `find`:

- | | |
|-----------------------------|------------------------------------------------------------|
| • <code>-type</code> | • <code>-and</code> , <code>-or</code> , <code>-not</code> |
| • <code>-name/-iname</code> | • <code>-maxdepth</code> |
| • <code>-size</code> | • <code>-print0</code> |
| • <code>-mtime</code> | • <code>-exec</code> |

Два вида синтаксиса `find -exec`:

- Синтаксис `c \;`

- Синтаксис `c +`

Чем различаются эти два вида синтаксиса? Примеры использования `find -exec`.

f. **Команда `xargs`**

Что делает команда `xargs`? В каких ситуациях эта команда используется чаще всего? Примеры использования команды `xargs`. Опции `-0`, `-I` и `-P`.

g. **Команда `tee`**

Примеры использования команды `tee`. Опция `-a`. Использование `tee` вместе с подстановкой процесса. Использование `echo`, `sudo` и `tee` для записи в файл с правами `root`.

h. **Команда `grep`**

Синтаксис команды `grep`. Опции:

- | | |
|---------------------------|-------------------------------------------------------|
| • <code>-i</code> | • <code>-c</code> |
| • <code>-r</code> | • <code>-o</code> |
| • <code>-n</code> | • <code>-w</code> |
| • <code>-v</code> | • <code>-A</code> , <code>-B</code> и <code>-C</code> |
| • <code>-E / egrep</code> | • <code>--color=auto</code> |
| • <code>-l</code> | • <code>-q</code> |

Приведите примеры использования `grep` в следующих ситуациях:

- Поиск строк, содержащих подстроку, в одном файле.
- Поиск строк, содержащих подстроку, рекурсивно во всех файлах директории.
- Поиск файлов, содержащих подстроку, рекурсивно во всех файлах директории.
- Фильтрация вывода команды с помощью конвейера и `grep`.
- Поиск строк в файле или файлах, соответствующих регулярному выражению.

i. **Регулярные выражения в `grep`**

Два типа регулярных выражений: BRE и ERE. Основные элементы регулярных выражений ERE:

- Обычные символы
- Точка `.`
- Символьные классы `[...]`
- Группировка `(...)`
- Альтернатива `|`
- Квантификаторы:

◦ <code>*</code>	◦ <code>{m}</code>
◦ <code>+</code>	◦ <code>{m,}</code>
◦ <code>?</code>	◦ <code>{m,n}</code>
- Якоря строки: `^` и `$`.

Предопределённые классы:

- | | |
|--------------------------|---------------------------|
| • <code>[:alpha:]</code> | • <code>[:lower:]</code> |
| • <code>[:digit:]</code> | • <code>[:upper:]</code> |
| • <code>[:alnum:]</code> | • <code>[:punct:]</code> |
| • <code>[:space:]</code> | • <code>[:xdigit:]</code> |

5. Пользователи и права доступа Linux

a. **Пользователи**

UID. Суперпользователь `root`. Системные пользователи. Зачем нужны системные пользователи? Какой диапазон UID обычно используется для системных пользователей, а какой для обычных?

b. **Группы пользователей**

Группы. Основная группа пользователя. Дополнительные группы. GID. Команда `id` и её опция `-u`. Команда `groups`.

c. **Домашняя директория**

Стандартное расположение домашней директории. Что обычно хранится в домашней директории? Файлы `~/.bash_profile` и `~/.bashrc`. Директория `/etc/skel`.

d. **Системные файлы, хранящие информацию о пользователях и группах**

- /etc/passwd
- /etc/shadow
- /etc/group
- /etc/gshadow

Какая информация и в каком формате хранится в каждом из этих файлов? Как хранится информация о паролях пользователей? Как хранится принадлежность пользователя к основной группе? Как хранится принадлежность пользователя к дополнительным группам? Как можно отличить обычного пользователя от системного по файлу `passwd`? Что такое `/sbin/nologin`? Как можно понять, что пользователь заблокирован по файлу `/etc/shadow`?

e. **Работа с пользователями и группами**

- `useradd` и её опции `-m`, `-d`, `-s` и `G`.
- `usermod` и её опции `-G`, `-aG`, `-d`, `-s`, `-L` и `-U`.
- `userdel` и её опция `-r`.
- `groupadd` и её опция `-r`.
- `groupmod`
- `groupdel`

Создание/удаление основной группы при создании/удалении пользователя. Использование `usermod` для блокировки/разблокировки пользователя.

f. **Работа с паролями**

Использование команды `passwd` для смены собственного пароля. Использование команды `passwd` для смены пароля другого пользователя.

g. **su и sudo**

Переключение на другого пользователя. Команда `su`. Полное и неполное переключение. Выполнение команды от имени другого пользователя. Команда `sudo` и её опции `-u` и `-i`. Какой пароль требуется для команды `su`, а какой пароль – для команды `sudo`? Чем различаются следующие способы переключения на пользователя `root`:

- `su`
- `su -`
- `sudo -i`

Файл `/etc/sudoers`. В каком формате хранятся записи в файле `/etc/sudoers`? Группа `sudo/wheel`. В чём опасность редактирования файла `/etc/sudoers`? Как безопасно редактировать этот файл?

h. **Права доступа**

Владелец файла. Группа владельца. Может ли владелец файла не входить в группу владельца этого же файла? Права доступа. Просмотр прав доступа с помощью `ls`. Символьное и числовое представление прав доступа. Конвертация из одного представления в другое. Команды:

- `chmod` и её опция `-R`. Как можно изменять права пользователя с помощью этой команды, используя символьное и числовое представление прав? Кто и в каких случаях может изменять права доступа?
- `chown` и её опция `-R`. Как изменить группу владельца, используя `chown`? Кто и в каких случаях может изменять владельца файла?
- `chgrp` и её опция `-R`. Кто и в каких случаях может изменять группу владельца файла?
- `newgrp`.

За что отвечают права `r`, `w`, `x` для обычных файлов? За что отвечают права `r`, `w`, `x` для директорий? Права мягких ссылок.

i. **Маска**

Права только что созданного файла. Права копии файла. Команда `umask`.

j. **SUID, SGID и Sticky Bit**

За что отвечает SUID? Примеры системных файлов, которые имеют SUID бит. Работает ли SUID на скриптах? За что отвечает SGID? SGID для обычных файлов и для директорий. За что отвечает Sticky Bit. Пример системной директории, которая имеет Sticky Bit. Символьное и числовое представление прав доступа вместе с данными битами.

6. Диски и файловые системы

- a. Единицы измерения информации
- b. Просмотр информации о дисках, разделах и точках монтирования
 - `lsblk`
 - `blkid`
 - `findmnt`
- c. Просмотр информации об использованной памяти
 - `df`
 - `du`
- d. Таблица разделов
MBR и GPT
- e. Создание разделов с помощью программы `parted`
 - `print`
 - `mklabel`
 - `mkpart`
 - `rm`
 - `resizepart`
 - `unit`

Запуск команд `parted` в скриптовом и тихом режиме. Команда `parted -l`.

- f. Работа с файловыми системами
Команда `mkfs`. Команды `mount` и `umount`. Файл `/etc/fstab`. Поля в этом файле `file system`, `mount point`, `type`, `options`, `dump`, `pass`. `UUID`.
- g. Команда `dd`
Распространённые варианты использования команды `dd`:
 - Создание файлов определённого размера, заполненный нулевыми или случайными байтами
 - Копирование файлов
 - Просмотр байт файлов
 - Просмотр байт разделов
 - Копирование разделов
 - Создание образа диска или раздела (файла `.img`)
 - Восстановление диска/раздела из образа
 - Запись ISO-образа на флешку

Опции команды `dd`:

- `if`
 - `of`
 - `bs`
 - `count`
 - `skip`
 - `seek`
 - `conv`, значения `noerror`, `sync`, `notrunc`, `ucase` и `lcase`
- h. Ссылки
Мягкие и жёсткие. Команда `ln` и её опция `-s`. Команды `ls -i` и `stat`. Зачем нужен счётчик жёстких ссылок в `inode`? Как узнать куда указывает мягкая ссылка? Что будет, если удалить исходный файл, на который указывает мягкая/жёсткая ссылка? Можно ли создать мягкую/жёсткую ссылку на директорию? Можно ли создать мягкую/жёсткую ссылку на файл, находящийся в другой файловой системе?
 - i. Файловые системы
Блоки. Суперблок. Таблица `inode`. Что хранится в `inode`-ах файлов разных типов? Переполнение таблицы `inode`.

- j. **Распространённые файловые системы**
ext4, xfs, btrfs, FAT, NTFS. Особенности каждой из файловых систем.

7. Язык Bash

- a. **Основы**
Интерпретатор Bash. Шелл.
- b. **Переменные Bash**
Создание и использование переменных в bash. Переменные среды. Команды `unset` и `export`. Команда `source`. Файл `.bashrc`.
- c. **Переменная среды PATH**
- d. **Аргументы**
Передача аргументов в Bash. Переменные `$1`, `$2`, `$0`, `$*`, `$@`.
- e. **Коды возврата**
Как получить код возврата последней выполненной команды?
- f. **Управляющая конструкция if fi**
Сравнение строк в конструкции `if`. Сравнение чисел в конструкции `if`. Операторы:
- `-n`
 - `-z`
 - `-e`
 - `-f`
 - `-d`
 - `-L`
 - `-r`
 - `-w`
 - `-x`
- Разница между `[[...]]` и `[...]`. Проверка на совпадение с регулярным выражением `=~`.
- g. **Управляющая конструкция case esac**
Примеры конструкций `case` `esac`. Использование `glob`-шаблонов в этих конструкциях.
- h. **Циклы**
Циклы `while` и `for`. Примеры циклов. Перенаправление из цикла и в цикл.
- i. **Манипуляции со строками**
Что означают следующие выражения:
- ```
$var
${var}
${#var}
${var:N}
${var:N:K}
${var:-default}
${var:=default}
${var#pattern}
${var%pattern}
${var/pattern/str}
${var^^}
${var,,}
```
- j. **Работа с целыми числами**  
Арифметические выражения `(( ... ))`. Чем `$(( ... ))` отличается от `(( ... ))`?
- k. **Функции**  
Создание функции. Вызов функции. Передача аргументов функции. Возврат значений из функции. Локальные переменные функции, чем они отличаются от обычных переменных?
- l. **Массивы**  
Уметь производить базовые операции с массивами: получение по индексу, изменение элемента, добавление элемента в конец массива.

m. **Чтение из стандартного входа или из файла**

Команда `read` и её опции `-p`, `-r`, `-a`, `-s`. Использование `read` вместе с `while` для чтения строк или слов из файла. Переменная `IFS`.

## 8. Процессы

a. **Основные определения**

Что такое программа? Что такое процесс?

b. **Управление заданиями**

Фоновый и передний режим. Как запустить процесс в фоновом режиме? Команды `jobs`, `fg`, `bg`. Использование `kill` для того, чтобы посылать сигналы процессам из `jobs`. Как остановить и возобновить процесс?

c. **Просмотр информации о процессах**

Команда `ps` и её опции `-e`, `-f`, `aux`, `-u`, `-o`. Команда `pstree`.

d. **Идентификаторы**

Идентификатор процесса `PID`. Как узнать идентификатор процесса? Как узнать идентификатор дочернего процесса? Группа процессов. Сессия.

e. **Сигнал**

Что такое сигнал? Основные сигналы:

- `SIGTERM`
- `SIGHUP`
- `SIGINT`
- `SIGKILL`
- `SIGSTOP`
- `SIGTSTP`
- `SIGCONT`
- `SIGCHLD`

Как послать сигнал процессу?

f. **Перехват сигналов**

Команда `trap`. Перехват сигналов. Игнорирование сигналов.

g. **Отсоединённый процесс**

Отсоединение процесса. Команда `nohup`.

h. **Состояния процессов**

- `Running`
- `Sleeping`
- `Uninterruptible sleep`
- `Stopped`
- `Zombie`

В каких случаях процесс попадает в то или иное состояние?

i. **Мониторинг процессов**

Команда `top` и `htop`. Директория `/proc`.

## 9. systemd

a. **Основы**

Что такое `systemd` и какую роль он выполняет в Linux? Юнит в `systemd`. Типы юнитов:

- `service`
- `target`
- `timer`
- `path`

b. **Команда `systemctl`**

Подкоманды:

- `systemctl start`
- `systemctl stop`
- `systemctl enable`
- `systemctl disable`
- `systemctl status`
- `systemctl cat`
- `systemctl reload`
- `systemctl daemon-reload`
- `systemctl list-units`

### c. Юнит-файлы

В каких директориях systemd ищет юнит файлы? Как создать свой сервис? Строение юнит файла. Секции. Директивы секции [Unit]:

- Description
- After/Before
- Wants/Requires
- Condition\*

Директивы секции [Service]:

- Type, типы `simple`, `forking`, `oneshot`, `notify`.
- ExecStart
- ExecStop
- Restart
- User/Group
- Environment/EnvironmentFile
- PIDFile
- TimeoutStartSec
- KillMode

Директива секции [Install]:

- WantedBy

Таргеты по умолчанию:

- `rescue.target`
- `multi-user.target`
- `graphical.target`
- `default.target`

### d. Таймеры

Зачем нужны timer юниты? Как создать timer unit, который бы запускал сервис по расписанию?

Директивы:

- Unit
- OnBootSec
- OnActiveSec
- Persistent
- RandomizedDelaySec
- OnCalendar

### e. Path-юниты

Зачем нужны path-юниты? Директивы:

- Unit
- PathExists
- PathChanged