
Linux Blogs/Sites:

https://www.cyberciti.biz/faq

https://linuxize.com

https://www.baeldung.com/linux/start-here

https://edmiachkov.notion.site/DevOps-Roadmap-2024-Rus-f2957674a2e74090b92ac5424d5db564

Restart computer:

```
$ reboot
```

---

show current (absolute) "navigation"-path in current terminal-tab/script/etc.:
command `pwd`:

```
$ pwd
```

example:

```
testusr@some-pc:~$ pwd
```

example output:

```
/home/testusr
```

---

how to `execute` (launch, start) some `binary file`:
(то есть: как стартануть-запустить "executable файл of some program"):

- Option 1:

Запустить бинарник по пути(по filePath-у):
- по абсолютному пути
- или по относительному пути

syntax:
```
$ <DirPath>/<FileName>
```
p.s. здесь обязательно должен указываться хотя бы один слеш
поэтому:
если файл прямо в текущей директории (в "$ pwd"), то в качестве директории указывается "точка" (.):
```
$ ./<FileName>
```
p.s в остальных случаях точка необязательна
Example:
если указываем абсолютный путь до файла
(p.s. абсолютный (absolute) путь - это full path of file)
то:
```
$ /home/testusr/c_lang_test/a.out
```
или так (с тильдой):
```
$ ~/c_lang_test/a.out
```
Sub-Example:
если файл в самой корневой директории of файловой системы (в "/"):
(p.s. например предварительно выполнили: "`$ sudo cp ~/c_lang_test/a.out /`")
то:
```
$ /a.out
```
Example:
если указываем относительный путь до файла
(p.s. относительный (relative) путь - это путь файла относительно of текущей директории (относительно of "$ pwd"))
то:
```
testusr@some-pc:~$ ./c_lang_test/a.out
```
или так (без точки):
```
testusr@some-pc:~$ c_lang_test/a.out
```

- Option 2:

в LinuxOS есть стандартная environmentVariable "PATH" (p.s. в WindowsOs тоже есть такая переменная окружения),

эта переменная содержит набор путей (разделенных через какой-то символ-разделитель),
так вот: если бинарник лежит в каком-то из этих путей, то тогда бинарник можно запускать просто "по названию файла":
syntax:

```
$ someBinary
```

example1:

```
$ cd
```

example2:

```
$ ls
```

example3:

```
$ example.bin
```

p.s. то есть при попытке запуска бинарника по названию, bash-terminal будет искать этот бинарник (с этим названием) во всех путях которые указаны в переменной PATH (p.s. поиск будет поочереди по каждому пути, и первый-"найденный"-"совпавший" бинарник будет запущен).

про CRLF:

CRLF и LF это разные концепции для "end of Line"

в Windows используется CRLF
в Linux-е используется LF

CR это "Carriage Return" = means: move the cursor to the beginning of the line
LF это "Line Feed" = means: move the cursor down to the next line

CR = 0x0D (HEX) (в hex формате (то есть в шестнадцатеричной системе счисления)), (то есть это в десятичной системе счисления = 13)
LF = 0x0A (HEX) (то есть это в десятичной системе счисления = 10)

CR в языках программирования обозначается таким выражением: '\r'
LF в языках программирования обозначается таким выражением: '\n'

Linux-Bash commands for: converting file between formats CRLF/LF:

- To convert from LF to CRLF:
  (p.s. execute в linux-bash terminal-e):

```
sed -i 's/$/\r/' fileLinux.txt
```

- To convert from CRLF to LF:
  (p.s. execute в linux-bash terminal-e):

```
tr -d '\r' <fileWindows.txt > fileLinux.txt
```

Linux `Users`:

- Create new user:

example:

```
$ sudo adduser u1
```

- Change password for user:

example:

```
$ sudo passwd u1
```

- Delete user:

example:
remove only user (not home-directory of user)

```
$ sudo userdel u1
```

example:
remove user and also user's-home-directory

```
$ sudo userdel -r u1
```

- Create new group:

```
$ sudo groupadd g1
```

- Add user to group

```
$ sudo adduser u1 g1
```

- List all groups of user:

(то есть: show groups a user is in (вывести список групп of юзера (список of групп в которых состоит пользователь))
syntax:

```
$ su <userName>
$ groups
```

or syntax:

```
$ groups <userName>
```

Example:

```
u1@some-pc:~$ groups
u1 users g1
u1@some-pc:~$ groups u1
u1 : u1 users g1
```

p.s. то есть у юзера "u1" 3 группы: "u1", "users", "g1".

Example:

```
testusr@some-pc:~$ groups
testusr users adm sudo <т.д.>
```

p.s. то есть у юзера "testusr" группы: "testusr", "users", "adm", "sudo".

Example:

```
testusr@some-pc:~$ groups root
root : root
testusr@some-pc:~$ sudo su
root@some-pc:# groups
root
```

p.s. то есть у суперпользователя (user "root") 1 группа: "root".

Example:

```
root@some-pc:/home/testusr# cat /etc/group
root:x:0:
adm:x:4:syslog,testusr,personal
sudo:x:27:testusr,personal
users:x:100:testusr,personal,postgres,u1,u2
ssl-cert:x:106:
testusr:x:1000:
postgres:x:1002:
g1:x:1003:u1,u2
u1:x:1004:
u2:x:1005:
<т.д.>
```

- List all groups (of OS):

```
$ getent group
$ getent group | grep <groupname>
```

- Remove user from group

```
$ sudo deluser u1 g1
```

- Delete group:

```
$ sudo groupdel g1
```

- p.s.

system-Linux-files которые обновляются (автоматически) при этих операциях:

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

---

command "usermod":

options:

-a, --append – добавить пользователя в одну или несколько дополнительных групп. Опция будет работать только вместе с опцией -G.
-b, --badnames – разрешить использование имен, которые не соответствуют стандартам.
-d, --home – указать новое местоположение домашнего каталога пользователя. При использовании опции -m содержимое текущего домашнего каталога будет перемещено на новое место.
-e, --expiredate – указать дату, при наступлении которой учетная запись будет отключена. Дата вводится в формате ГГГГ-ММ-ДД. Если использовать эту опцию без указания даты, то отключение пользователя будет отменено.
-f, --inactive – установить количество дней для блокировки пользователя, которое должно пройти после устаревания пароля. При значении -1 опция блокировки отключается, а при значении 0 блокировка случится сразу же после устаревания.
-g, --gid – выбрать новую основную группу для пользователя и для файлов в его домашнем каталоге. Нужно задать имя или номер новой группы.
-G, --groups – указать список дополнительных групп, в которые должен входить пользователь. Между собой группы разделяются запятой. Если пользователь входит в дополнительную группу, которая не была указана в списке, то он будет из нее удалён. Но при использовании опции -a можно добавлять новые дополнительные группы, не удаляя старые.

-l, --login – изменить имя пользователя на новое. Данная опция не затрагивает никакие другие данные. А значит, название домашнего каталога и почты придется изменять вручную, чтобы они совпадали с новым именем пользователя.

-L, --lock – заблокировать пароль пользователя. Эта опция помещает символ ! (восклицательный знак) перед паролем в зашифрованном виде, отключая его. Данную опцию нельзя использовать с -p и -U.

-m, --move-home – изменить местоположение домашнего каталога пользователя. Опция будет работать только вместе с -d. Утилита попытается обновить права собственности на файлы и скопировать режимы, ACL и расширенные атрибуты.

-o, --non-unique – разрешить заменить идентификационный номер пользователя на не уникальное значение. Работает в паре с опцией -u.

-p, --password – изменить пароль в зашифрованном виде.

-R, --root – выполнить chroot в указанный каталог и использовать его вместо корневого каталога / с хранящимися в нем конфигурационными файлами.

-s, --shell – указать новую командную оболочку shell для пользователя. При использовании опции -s с пустым значением будет выбрана оболочка по умолчанию.

-u, --uid – изменить параметр UID (числовой идентификатор пользователя). Данные изменения автоматически применятся к почтовому ящику и содержимому домашнего каталога. Для остальных файлов UID придется изменять вручную.

-U, --unlock – разблокировать пароль пользователя. Данная опция убирает символ ! (восклицательный знак) перед паролем в зашифрованном виде, разрешая использовать его для входа. Не сработает с -p и -L.

Examples:

- Изменить основную группу

Посмотреть список всех групп, доступных в системе, можно в файле /etc/group, например, с помощью редактора vi в терминале:
vi /etc/group

Для смены основной группы нужна опция -g. Синтаксис здесь следующий:

$ usermod -g имя_основной_группы имя_пользователя

Задача – изменить основную группу для пользователя test_user на test_group (GID – 1001). Так будет выглядеть команда в нашем случае:
sudo usermod -g test_group test_user

Затем можно проверить что изменения применились с помощью команды id. В результатах вывода команды id нам интересен пункт GID. А еще вместо названия группы можно использовать ее идентификатор GID (1001 в нашем случае):
sudo usermod -g 1001 test_user

- Добавить пользователя gregory2 в группу plugdev

С этой задачей поможет параметр -G. Но его обязательно нужно использовать вместе с -a, чтобы добавить новую группу, не удаляя старые:

sudo usermod -a -G plugdev gregory2

- т.д.

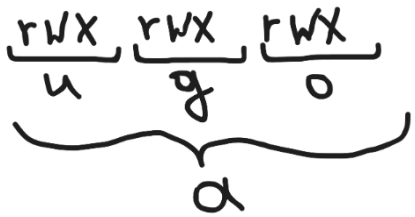
links:
- https://losst.pro/komanda-usermod-v-linux
- https://linuxize.com/post/usermod-command-in-linux

---

file/dir `Permissions` in Linux:

```
$ man chmod
```

ugoa



u это **user** who owns file (то есть: owner of file)
g это other users in the file's **group**
o это **other** users которые - not in the file's group
a это **all** users (т.е. применится к [u, g, o])

- r это read (чтение, просмотр)

- for file:

view content of file
(p.s. command "cat", or open in editor (for readonly), т.д.)
(p.s. Read permission is required to make copies of a file, because you need to access the file's contents to make a duplicate of it.)

- for dir:

view contents of directory (p.s. commands "ls", т.д.)

- w это write (modify, изменять контент файла)
- for file:

change content of file (write/append).

- for dir:

change contents of directory (add file ("touch"), delete file ("rm"), move file ("mv"), copy file ("cp")).

- x это execute (запустить)
- for file:

execute content of file (launch binary-file or shell-script).

- for dir:

Execute permission on a directory authorizes you to look at extended information on files in the directory (using ls -l, for instance), and also allows change current-directory (command "cd" to directory or to subdirectory).

- Числовые-значения:

r это 4
w это 2
x это 1
(всё вместе) = 7 (то есть это сумма: 4+2+1)
(а только read+write =4+2=6)
(а только read+execute =4+1=5)
(а только write+execute =2+1=3)

- p.s.

чтобы выдать разрешение на файл "someDir/someSubDir/someFile"
нужно выдать разрешение И на файл И на все-подпути этого файла "рекурсивно"
(то есть: И на "someDir/someSubDir/someFile", И на "someDir/someSubDir", И на "someDir")
иначе будет error: permission denied (например если выдали разрешение только на файл, А для subdir-a/dir-a (of этого файла) например readPermission Не выдали, в итоге логично что будет запрещен доступ к файлу так как запрещен доступ к subdir-y/dir-y of этого файла).
Example:

```
testusr@some-pc:~$ echo <someSqlQueries> > /home/testusr/d1/sd1/f.sql

testusr@some-pc:~$ \
chmod +rwx /home/testusr/d1/sd1/f.sql; \
chmod +rwx /home/testusr/d1/sd1; \
chmod +rwx /home/testusr/d1; \
chmod +rwx /home/testusr;

testusr@some-pc:~$ su - postgres
postgres@some-pc:~$ psql -d db1 -f /home/testusr/d1/sd1/f.sql
```

p.s. то есть в этом примере пользователю postgres будет разрешен доступ на выполнение sql-файла "f.sql"

Grant Permissions to file/dir:
(выдать права(разрешения) на file/dir)
command `chmod`:

example:

```
$ chmod a+r someFilePath
```

p.s. означает выдать всем пользователям (a = all) право на чтение (r = read) (чего?) файла (path = someFilePath)

example:

```
$ chmod o-w someFilePath
```

p.s. означает запретить другим пользователям (o = others) изменять/modify (w = write) файл (path = someFilePath)

example:

```
$ chmod u+x someFilePath
```

означает добавить владельцу файла (u = user) право/permission на выполнение/запуск (x = execute) файла (path = someFilePath)

example:
p.s. если Не указывать u/g/o/a перед символом (+/-/=) То тогда это будет применяться ко всем юзерам (ugoa) кроме суперпользователя (root).

```
$ chmod -rwx someFilePath
```

p.s. означает запретить Любому пользователю (ugoa, кроме суперпользователя) Любую операцию (rwx) над файлом (path = someFilePath)

т.д. (другие примеры по аналогии)

meanings:
+ это add permission
- это delete permission
= это назначить permission

Change `ownership` of file/dir:
command `chown`, `chgrp`:

- Change owner of file/dir:

syntax:
```
chown <UserWhoWillBeNewOwner> <DirPathOrFilePath>
```
example:
```
chown u1 /home/u1/f.txt
```
p.s. for example: old owner was "root", new owner is "testusr"

- Change group of file/dir:

syntax:
```
$ chgrp <GroupToSet> <FilePathOrDirPath>
```
example:
```
$ chgrp g1 /home/u1/f.txt
```

- Change user and group of file/dir:

syntax:
```
$ chown <UserWhoWillBeNewOwner>:<GroupToSet> <FilePathOrDirPath>
```
example:
```
$ chown u1:g1 /home/u1/f.txt
```

---

command `whoami`:
```
testusr@some-pc:~$ whoami
```

```
testusr
```

p.s. то есть: current user: "testusr"

про юзера "root":

- директория этого пользователя находится в корне of файловой системы, а именно: путь равен: **/root**

юзер root эти типа "суперадминистратор" (то есть Не просто какой-то из администраторов, А "суперадминистратор" of операционной системы (суперадминистратор of текущего **компьютера**) (то есть такой "суперадминистратор" только один на весь текущий компьютер))

как сменить "текущего" пользователя в текущей terminal-вкладке: command `su`:

- Option 1:

syntax:

```
$ su <UserToSwitch>
```

p.s. в этом варианте: директория of предыдущего юзера остается в качестве текущей директории для <UserToSwitch>

Example:

```
u1@some-pc:~$ pwd
/home/u1
u1@some-pc:~$ su u2
Password:
u2@some-pc:/home/u1$ pwd
/home/u1
```

- Option 2:

syntax:

```
$ su - <UserToSwitch>
```

p.s. после su написан "дефис"

p.s. в этом варианте: директория of предыдущего юзера Не остается, а ставится на дефолтную /home/<UserToSwitch>

p.s. можно запомнить это так: что как будто дефис это знак минус и как будто вычитаем (exclude-им, игнорируем) директорию of предыдущего юзера.
Example:

```
u1@some-pc:~$ pwd
/home/u1
u1@some-pc:~$ su - u2
Password:
u2@some-pc:~$ pwd
/home/u2
```

- Option 3:

сменить на: суперпользователя (of операционной системы)
p.s. суперпользователь = user "root"
p.s. Но чтобы сменить(switch) "текущего юзера" в terminal-tab-е на суперпользователя нужно чтобы "текущий юзер" был администратором (состоял в группе администраторов), иначе запрещено switch-ится от "Не администратора" на суперпользователя.
Example:

```
testusr@some-pc:~$ pwd
/home/testusr
testusr@some-pc:~$ sudo su
[sudo] password for testusr:
root@some-pc:/home/testusr# cd ~
root@some-pc:~# pwd
/root
```

как выполнить команду от имени именно "суперадминистратора" (то есть от user-а "root"):

command `sudo`:
означает: выполнить команду **именно от** пользователя **root**:
Syntax:

```
sudo <SomeCommand>
```

Example:

```
sudo ls -R /*
```

p.s. то есть **будто** выполнили так:

```
$ sudo su
$ <SomeCommand>
```

p.s. в разных программах/приложениях по разному реализовано “выполнение команды “sudo”
например в UbuntuGraphicFileExplorer-е (в конкретной открытой текущей **instance-e**) при попытке открыть “директорию другого пользователя /home/<AnotherUser>) появится модальное окошко с вводом пароля (of root user) - если там ввести пароль - директория откроется,
а например если просмотр файловой системы через терминал:
например через утилиту lf
то можно сразу запустить утилиту “от рута” (то есть: $ sudo lf)
и тогда (в этой terminal-вкладке в этой программе lf) будет доступ к директориям которые доступны от суперпользователя (например будут доступны локальные директории of других юзеров)

---

navigate to some directory within terminal:
command `cd`:
Syntax:

```
cd <dirPath>
```

Example:

```
testusr@some-pc:~/Data$ cd ~/Downloads
testusr@some-pc:~/Downloads$
```

---

show man page (documentation) for some command:
Syntax:

```
man <commandName>
```

Example:

```
testusr@some-pc:~/Data$ man ls
```

p.s. откроется man page
p.s. для "закрытия" of man страницы: нажать `q`
p.s. иногда альтернативная команда такая: `<cmd> --help` (example: `ls --help`)
или такая: `info <cmd>`
p.s. также можно прочитать man страницу в интернете https://manpages.ubuntu.com/manpages (в правом верхнем углу of этого website-а

есть "поисковая строка" - там написать название команды которую нужно найти (например: "ls")

---

create directory:
Options/flags:
-p means: also create parent directories - if parent directories not exist
Example:

```
mkdir TestDir
cd TestDir
mkdir d1
mkdir d1/sd1
mkdir d1/sd2
mkdir d2
```

Example:

```
mkdir -p ~/NotYetExist/SomeSubDir/MostNestedSubdir
```

---

create file:

```
cd TestDir
touch d1/sd1/f1
touch d2/f2
```

---

write to some file:
command `echo`:
Syntax Description:
> means write (same to override)
>> means append (to the end of file)
Examples:

```
testusr@some-pc:~$ echo "1 f" > ~/TestDir/d1/sd1/f1
testusr@some-pc:~$ echo "2 f" > ~/TestDir/d2/f2
```

Неочевидный момент:
команда echo Неявно добавляет в конец файла: "системный символ" "LF" (0x0A)
то есть если открыть в hex-editor-е файл `~/TestDir/d1/sd1/f1`
то такой контент:

```
31 20 66 0A
```

p.s.
31 = "1" (цифра один)
20 = " " (пробел)
66 = "f" (буква f)
0A = Enter (символ "Enter" = это символ обозначает что началась "New Line")
p.s. если потом выполнить команду

```
testusr@some-pc:~$ echo "z" >> ~/TestDir/d1/sd1/f1
```

то в hex-editor-е будет такой контент:

```
31 20 66 0A 7A 0A
```

p.s. 7A = "z" (буква z)

---

count bytes/lines/etc in some file:
command `wc`:

syntax:

```
$ wc <options> <someFile>
```

options:
`-l`: количество of всех строк(lines) in a file.
-w: количество of всех отдельных-слов(words) in a file.
-c: количество of всех байтов(bytes) in a file.
-m: количество of всех символов(characters) in a file (p.s. учитывая(включительно) символ LF).
-L: количество of символов(characters) in the longest line in a file (в самой длинной строке of файла) (p.s. Не учитывая(Не включительно) символ LF).

example:

```
$ echo "zfs Д" > f.txt

$ xxd f.txt
7a 66 73 20 d0 94 0a
```

p.s. 7a=z, 66=f, 73=s, 20=<whitespace>, [d0 and 94]=Д, 0a=<LF>

```
$ wc -l f.txt
1 f.txt

$ wc -w f.txt
2 f.txt
```

```
$ wc -c f.txt
7 f.txt

$ wc -m f.txt
6 f.txt
```
p.s. z, f, s, <whitespace>, Д, <LF>

```
$ wc -L f.txt
5 f.txt
```
p.s. z, f, s, <whitespace>, Д

```
$ wc f.txt
1 2 7 f.txt
```
p.s. <-l> <-w> <-c> <filePath>

---

clear all text on current terminal-tab-screen:
command `clear`:
```
testusr@some-pc:~$ blabla
testusr@some-pc:~$ clear
```
result:
```
testusr@some-pc:~$
```

---

copy file/dir to another dir:
command `cp`:
Options/flags:
-r означает разрешение-подтверждение что копировать вложенные nested sub-directories
Example:
```
testusr@some-pc:~$ cp -r /media/testusr/KINGSTON/OTHER ~/FlashcardCopy/
```

---

show/view size of file/directory:
command `du`:
Options/flags:

-b = show size in `bytes` (Если этот флаг Не указать - то почему-то выводятся Совсем Некоректные значения!)
-s = total size of directory (Если этот флаг Не указать - то будет выводится информация рекурсивно для всех nested директорий/файлов, А если указать - то выводится только одна line обобщенная для указанной директории- (total size of directory)
-h = human readable format ((KB) Kilobytes, (MB) Megabytes, т.д. - вместо xxxxBytes) (Если этот флаг Не указать то будет выводиться число которое обозначает количество байтов, А если указать этот флаг - то будет выводится сокращенная запись (то есть если количество байт бОльше чем 1024 то будет отображаться сокращенная запись <x>K (x Kilobytes) (если бОльше чем 1024*1024 то будет отображаться <x>M (x Megabytes), т.д.

Example:

```
testusr@some-pc:~/Data$ du -bh /home/testusr/TestDir
3       /home/testusr/TestDir/d2
252     /home/testusr/TestDir/d1/sd1
0       /home/testusr/TestDir/d1/sd2
252     /home/testusr/TestDir/d1
255     /home/testusr/TestDir
testusr@some-pc:~/Data$ du -bsh /home/testusr/TestDir
255     /home/testusr/TestDir
testusr@some-pc:~/Data$ du -bsh /usr/local/go
227M    /usr/local/go
```

p.s. еще удобно при копировании (cp) проверять сколько уже скопировалось и какое должно быть итоговое количество байт после того как всё скопируется

Example:

```
testusr@some-pc:~$ du -bs ~/FlashcardCopy/
20258228155 /home/testusr/FlashcardCopy/
testusr@some-pc:~$ du -bs /media/testusr/KINGSTON/OTHER/
52330103541 /media/testusr/KINGSTON/OTHER/
testusr@some-pc:~$ du -bs ~/FlashcardCopy/
52330103541 /home/testusr/FlashcardCopy/
```

Example:

```
testusr@some-pc:~/Documents$ du -bh
451   ./Obsidian Vault/.obsidian/snippets
317K  ./Obsidian Vault/.obsidian/plugins/janitor
19K   ./Obsidian Vault/.obsidian/plugins/settings-search
336K  ./Obsidian Vault/.obsidian/plugins
342K  ./Obsidian Vault/.obsidian
11K   ./Obsidian Vault/.trash
355K  ./Obsidian Vault
355K  .
```

```
testusr@some-pc:~/Documents$ du -sbh
355K
```

---

rename (move) file:
command `mv`:

```
mv <OldFilePath> <NewFilePath>
```

Example:
rename file:

```
mv ~/TestDir/f_old ~/TestDir/f_new
```

p.s. result: only ~/TestDir/f**_new**
Example:
move file to another dir:

```
mv ~/TestDir/f1 ~/NewTestDir/
```

p.s. result: only ~/**New**TestDir/f1

---

print content of file:
command `cat`:
Example:

```
cat ~/TestDir/f1
```

---

command `more`:
(almost same as command "less")
Example:
view only one file:

```
more ~/TestDir/f1
```

Example:
view all files of dir (sequentially):

```
more ~/TestDir/*
```

p.s. same hotkeys as for "less"

---

view content of file (or several files in dir) per page size:
command `less`:

Example:
view only one file:
```
less ~/TestDir/f1
```
Example:
view all files of dir (sequentially):
```
less ~/TestDir/*
```

hotkeys:

- PageUp/PageDown - to scroll current file per page size,
- Up/Down - to scroll current file per line,
- Home/End - to navigate to start/end of page
  (p.s. да - в этой утилите "Home"/"End" обозначают "другое": то что обычно означают "Ctrl+Home"/"Ctrl+End")
  (p.s. команды "`:n`" и "`:p`" Необязательно открывают файл с начала, так что лучше после команд "`:n`"/"`:p`" нажимать еще явно "Home"/"End")
- "`:n`" - for navigate to **n**ext file (in dir)
- "`:p`" - for navigate to **p**revious file (in dir (p.s. если "less" for dir))
- "`:f`" - to display current FilePath
- "`q`" (or "ctrl+z") - to quit from "less-editor" and return to terminal-mode

(p.s. https://linuxize.com/post/less-command-in-linux/)

---

view content of file with some features:
command `tail`:
Example:
```
workusr@another-pc:~$ tail -F ~/TestDir/f1
```
p.s. -F именно большой буквой (not f), так как тогда будет мониториться именно по названию файла (A Не по текущему экземпляру of файла) - так как если удалить файл а затем создать заново файл с таким же названием (по тому же пути) (но с каким-то новым контентом) то если флаг -F то изменения замониторятся (так как мониторинг будет по названию файла (A Не по конкретно этому экземпляру of файла)) (A если флаг -f то тогда tail будет мониторить конкретно этот экземпляр (копию) of файла (который останется на диске пока запущены процессы которые используют эту "старую" копию (то есть например пока будет запущен процесс tail))
p.s.
https://unix.stackexchange.com/questions/379982/editing-text-file-with-vim-does-not-update-tail-f

p.s. output of example:

```
<initial version of file (initial_content)>
<updated version of file (updated_content)>
<latest version of file (latest_content)>
```

p.s. output будет автоматически "дополняться" - сначала будет выведен изначальный текущий контент of file, затем (когда файл будет обновлен в сторонней программе) то ниже будет автоматически выведен весь "новый текущий" контент of файла, т.д.

---

show filesystem-structure (and properties) of some directory (or for file) as tree:
command `tree`:

```
apt install tree
```

Options/flags:
-a = print all files (including files which starts with dot (example: .gitignore, so on))
-d = print only directories
-f = print full path (for each item)
-L <N> = limit deep-depth for displaying (p.s. вроде как по умолчанию команда tree отображает все nested sub-директории (то есть типа весь deep-depth) (а чтобы наоборот ограничить output до какого-то deep-уровня то задать опцию -L <N>, где N это max deep level для отображения в output-е)
-R = рекурсивный вывод (p.s. непонятно зачем эта опция, так как по дефолту всё рекурсивно выводится)
Example:

```
testusr@some-pc:~$ tree ~/TestDir
/home/testusr/TestDir
├── d1
│   ├── sd1
│   │   └── f1
│   └── sd2
└── d2
    └── f2

5 directories, 2 files
```

Example:

```
testusr@some-pc:~$ tree ~/TestDir -L 1
/home/testusr/TestDir
├── d1
```

```
└── d2

3 directories, 0 files
```

Example:

```
testusr@some-pc:~$ tree -d ~/TestDir
/home/testusr/TestDir
├── d1
│   ├── sd1
│   └── sd2
└── d2

5 directories
```

Example:

```
testusr@some-pc:~$ tree -f ~/TestDir
/home/testusr/TestDir
├── /home/testusr/TestDir/d1
│   ├── /home/testusr/TestDir/d1/sd1
│   │   └── /home/testusr/TestDir/d1/sd1/f1
│   └── /home/testusr/TestDir/d1/sd2
└── /home/testusr/TestDir/d2
    └── /home/testusr/TestDir/d2/f2

5 directories, 2 files
```

Example:

pipe with grep

```
workusr@another-pc:~/Downloads/glibc-2.0.6$ tree -f | grep -i "stdio.c"
│   ├── ./libio/stdio.c
│   ├── ./stdio/clearerr.c
├── ./stdio-common
│   ├── ./stdio-common/asprintf.c
```

p.s. так как точка (.) в `grep -i "stdio.c"` Незаescape-ена то тогда эта точка означает любой символ (например дефис, слеш, и т.д.) (так как в grep-е указывается именно regexp выражение)

```
workusr@another-pc:~/Downloads/glibc-2.0.6$ tree -f | grep -i "stdio\.c"
│   ├── ./libio/stdio.c
│   └── ./sunrpc/xdr_stdio.c
│   │   ├── ./sysdeps/generic/sysd-stdio.c
```

p.s. а здесь уже точка (.) Заescape-ена

```
workusr@another-pc:~/Downloads/glibc-2.0.6$ tree -f | grep -i "/stdio\.c"
|   ├── ./libio/stdio.c
```

p.s. а здесь еще "сказано" что перед "stdio" должен быть символ слеш (/) (то есть типа что "stdio" это начало of названия какого-либо файла/директории (на любом уровне вложенности)

---

show filesystem-structure (and properties) of some directory (or for file)
as list:
command `ls`:
Options/flags of command:
`-1` = each output item on separate line (output per line)
`-R` = deep recursively (output all nested subfolders)
`-l` = long listing format (file permissions, creation date, т.д.)
-a = add to output hidden files/folders (that starts with dot . (.gitconfig, . (link to current directory, .. (link to parent directory), т.д.))
-s = show file size (Only for files)
p.s. в самой левой колонке отображается Неправильное значение (это такое значение которое означает: "on disk" (отображается ближайшая округленная в бОльшую сторону степень of 4-х KB-байт (так как реальный диск считывает по блокам (размер блока 4KB) - то есть типа отображается: сколько будет прочитано блоков of жесткого диска)))
А в дальней правой колонке отображается правильное значение: точный размер of файла (НО для директорий отображается Неправильное значение - там типа всегда просто дефолтное значение отображается 4KB (а вот команда `du` выводит правильно для директорий, а вот файлы наоборот не выводит, так что size of files смотреть через утилиту `ls`, а size of directories смотреть через утилиту `du`)
-h = human readable format of file size (KB/MB/т.д.)
-S = sort by file size (largest at top)
-t = sort by last modified time
-d = вывести только один указанный-в-аргументе file/dir (А вложенные Не будут выведены)
-f = вывести плоский-горизонтальный список (А Не вертиальный) (будут выведены только первый уровень of файлов/директорий, а вложенные Не будут выведены).
p.s. остальные опции в man-е читать если что, а то слишком много опций
Syntax:
ls <options> <DirPath**Or**FilePath>
p.s. то есть можно выполнять команду И для директории И для конкретного файла (например чтобы посмотреть permissions, т.д. только для конкретного файла)

Example:

```
ls ~/TestDir
```

Example:

```
ls -l ~/TestDir
```

Example:

```
ls -l ~/TestDir/d1/sd1/f1
```

Example:

```
testusr@some-pc:~$ ls -RshlaS ~/Documents/ObsidianVault
# examples of output:
4.0K -rw-rw-r-- 1 testusr testusr 1.5K Dec  3 08:00 styles.css
12K  -rw-rw-r-- 1 testusr testusr  11K Dec  3 08:02 someImage.png
4.0K -rw-rw-r-- 1 testusr testusr  589 Dec  3 05:00 styles1.css
4.0K -rw-rw-r-- 1 testusr testusr  296 Dec  3 05:00 manifest1.json
4.0K -rw-rw-r-- 1 testusr testusr 1.5K Dec  3 08:00 styles2.css
4.0K -rw-rw-r-- 1 testusr testusr  310 Dec  3 08:00 manifest2.json
316K -rw-rw-r-- 1 testusr testusr 315K Dec  3 08:00 main.js
```

Example:

```
testusr@some-pc:~$ ls -1 ~/TestDir
d1
d2
```

Example:

```
testusr@some-pc:~$ ls -R ~/TestDir
/home/testusr/TestDir:
d1  d2

/home/testusr/TestDir/d1:
sd1  sd2

/home/testusr/TestDir/d1/sd1:
f1

/home/testusr/TestDir/d1/sd2:

/home/testusr/TestDir/d2:
f2
```

Example:

```
testusr@some-pc:~$ ls -alR -1 ~/TestDir
/home/testusr/TestDir:
total 16K
4.0K drwxr-x--- 23 testusr testusr 4.0K Nov 30 13:14 ..
4.0K drwxrwxr-x  4 testusr testusr 4.0K Nov 30 13:14 .
4.0K drwxrwxr-x  4 testusr testusr 4.0K Nov 30 13:12 d1
```

```
4.0K drwxrwxr-x  2 testusr testusr 4.0K Nov 30 13:12 d2

/home/testusr/TestDir/d1:
total 16K
4.0K drwxrwxr-x 4 testusr testusr 4.0K Nov 30 13:12 .
4.0K drwxrwxr-x 4 testusr testusr 4.0K Nov 30 13:14 ..
4.0K drwxrwxr-x 2 testusr testusr 4.0K Nov 30 13:12 sd1
4.0K drwxrwxr-x 2 testusr testusr 4.0K Nov 30 13:12 sd2

/home/testusr/TestDir/d1/sd1:
total 8.0K
4.0K drwxrwxr-x 2 testusr testusr 4.0K Nov 30 13:12 .
4.0K drwxrwxr-x 4 testusr testusr 4.0K Nov 30 13:12 ..
  0 -rw-rw-r-- 1 testusr testusr    0 Nov 30 13:12 f1

/home/testusr/TestDir/d1/sd2:
total 8.0K
4.0K drwxrwxr-x 2 testusr testusr 4.0K Nov 30 13:12 .
4.0K drwxrwxr-x 4 testusr testusr 4.0K Nov 30 13:12 ..

/home/testusr/TestDir/d2:
total 8.0K
4.0K drwxrwxr-x 2 testusr testusr 4.0K Nov 30 13:12 .
4.0K drwxrwxr-x 4 testusr testusr 4.0K Nov 30 13:14 ..
  0 -rw-rw-r-- 1 testusr testusr    0 Nov 30 13:12 f2
```

Example:

```
root@some-pc:~# ls -lad /home/u1
d--------- 3 u1 u1 4096 Mar  9 19:37 /home/u1
```

Example:

```
root@some-pc:~# ls -laf /home/u1
dir1  .profile  .bash_history  .bash_logout  ..  .bashrc  .
```

---

copy file to some directory:
command `cp`:
cp <SourceFile> <DestinationDirectory>
Example:

```
$ sudo cp ~/Downloads/SomeFile /usr/local
```

as result: создался файл-"копия": `/usr/local/SomeFile`

---

show history of commands that were entered to terminal:
command `history`:

```
testusr@some-pc:~$ history
   1  sudo apt-get update
   2  ls
   3  dpkg -i ./some.deb
   4  ls
   5  mkdir -p ~/TestDir
```

---

remove directory:
command `rm`:
Example:

```
testusr@some-pc:~$ rm -rf ~/TestDir
```

p.s. флаг **`-r`** означает recoursevely (рекурсивно) (то есть этот флаг - это "подтверждение-разрешение" что удалять вложенные (nested) директории которые внутри этой директории),
флаг **`-f`** означает force (форсированно) (то есть этот флаг - это "подтверждение-разрешение" что удалять директорию даже если эта директория используется в каком-то из запущенных процессов (приложениях)
Example:
чтобы удалить весь контент внутри директории (Но не удалять директорию (~/TestDir)):
то такая команда:

```
$ rm -rf ~/TestDir/*
```

p.s. на конце именно тОлько * (только звездочка (после слеша))
(p.s. а если написать /.* (слеш, точка, звездочка) - То ничего Не произойдет (Ничего Не удалится)
Example:
чтобы удалить несколько файлов по regexp-у:
(p.s. предварительно удостовериться что никакие лишние файлы не попадают под этот regexp, p.s. на проде наверно вообще лучше не использовать эту команду, но для локальных тестов - это удобно - когда точно знаешь какие тестовые файлы были созданы и что шаблон (regexp) затронет только конкретно точно известные файлы)

Example:
rm ~/Data/***tar***
p.s. например эта команда удалит 3 файла: TestDir.**tar**, TestDir.**tar**.gz, TestDir.**tar**.bz2
p.s. Но в rm работают Не все regexp-ы (по крайнем мере по "простому" escap-ингу как работает в других командах (например в "find"-е)) и также: как уже написала выше: Не правильно сразу удалять по regexp-у А рекомендуется удалять точно известное конкретное название файла то есть сначала "найти" файл по regexp-у (например: $ find -name "**some***(1)*"") (например результат: ./**some**DublicateFile**(1)**.deb) а затем уже удалить по этому конкретному "найденному" названию ($ rm ./**some**DublicateFile**(1)**.deb)
p.s. команда rm Не кладет удаляемый файл/директорию в Trash/RecycleBin (по факту: команда rm тОлько удаляет ссылку (link, reference) на файл (ссылка на область памяти где хранится файл) (а контент файла остается в памяти, но ссылка-"привязка"-"маппинг" удаляется => то есть этот блок памяти типа marked (помечен) что Не используется (то есть разрешен для allocate-инга (то есть потом новые данные просто Перезатрут-поверх (override) эти существующие данные в этом блоке памяти))

---

execute several commands:

- **последовательно:**

syntax:
```
<Command_1> && <Command_2> && <Command_3> т.д.
```
p.s. сначала выполнится (полностью, до конца) команда Command_1, затем После того как Command_1 завершится - начнет выполняться команда Command_2, затем После того как завершится команда Command_2, начнет выполняться команда Command_3, т.д.
Example:
```
$ ls && mkdir d1 && rm d1 && ls
```
- p.s. ДОПИСАТЬ в чем разница с "через точку-с-запятой":

```
$ ls; mkdir d1; rm d1; ls;
```

- **одновременно:**

syntax:
```
<Command_1> & <Command_2> & <Command_3> т.д.
```

p.s. запустится команда Command1, затем сразу-же запустится команда Command2 (не дожидаясь завершения of Command1), затем сразу запустится Command3, т.д. То есть команды будут выполняться одновременно.
Example:
```
echo "random order of printing" & echo "test"
```

"Pipe" (redirect, перенаправить) output of some command:

- **To -> input of another command**

example:
```
curl https://some.sh | powershell
```
то есть some.sh это “input” для powershell terminal-a
(то есть дословно это преобразуется в следующее:

1. выполнить скачивание скрипта (`curl https://some.sh`)
2. запустить программу `powershell` (в background-е ("в фоновом режиме"), то есть окно терминала Не будет появляться вроде)
3. передать программе powershell на вход - результат **1-й команды** (то есть: скачанный скрипт) - то есть Powershell получит на вход скрипт `some.sh` (**=** то есть набор команд (которые написаны в этом скрипте `some.sh`) (**будто** мы вручную последовательно вводим эти команды в powershell терминал)

- **To -> some file:**

command `>`:
```
testusr@some-pc:~$ git help config > SomeOutputFile
```

---

command `tail`:
вывести <N> строк с конца (latest N lines from the end (insted of from start)):
Syntax:
<SomeCommandWillBeInputForTail> | tail -n <N>
Example:
testusr@some-pc:/usr/local/go**$** history | **tail -n 10**

---

show only те output lines которые содержат конкретные указанные символы/слова/regex-паттерны:

command `grep`:
Syntax:
<SomeCommandWillBeInputForGrep> | grep <SomeRegExp>
Example:

```
ls | grep ".h"


ls | grep '.*.cs'
```

Example:
Не фильтровать output - А только выделить цветом конкретные символы:

```
$ ls -R ~/TestDir | grep f*
```

example of output:

```
/home/testusr/TestDir/d1/sd1:
f1


/home/testusr/TestDir/d1/sd2:


/home/testusr/TestDir/d2:
f2
```

p.s. То есть:
вывелось **Всё** (то есть всё что было в команде ls),
НО просто **Выделилось** (красным цветом): **то** что было задано в команде **grep** (то есть то что удовлетворяет патерну `f*`)
Example:
Search of some text in multiple files:
(например поиск слова "app" во всех файлах которые внутри директории "/usr/share/applications")

```
testusr@some-pc:~$ grep -ir 'app' /usr/share/applications
```

p.s.
-i значает игнорирование of регистра (то есть найдет и "app" и "App" и "aPp" и т.д.)
r - означает что поиск рекурсивный - То есть для всех файлов которые внутри директории И во вложенных sub-директориях (то есть найдет и в `/usr/share/applications` и в `/usr/share/applications/subDir` и т.д.)
p.s. output = список of filepathes of найденных (matched) файлов
Example:
поиск нескольких значений через "ИЛИ":

```
grep -ir 'word1|word2|word3' '/someDir'
```

Example:
grep for binary file:

```
$ grep -i Hello ~/a.out
```

output:

```
grep: /home/testusr/a.out: binary file matches
```

Example:

grep for binary file:

```
$ xxd ~/a.out | grep -i Hello
```

output:

```
00002000: 0100 0200 4865 6c6c 6f2c 2077 6f72 6c64  ....Hello, world
```

про TAR archive:
(про create/extract `tar` archive):

tar archive - это такой архив который Не сжатый (not compressed), первоначальная цель tar архивов это для удобной пересылки Нескольких файлов по сети, то есть несколько файлов типа "комбнировали" в один единый специальный файл: tar архив (там файлы внутри Не сжатые)
А уже **tar.gz** и **tar.bzip2** это такие архивы - внутри которых: оптимизированно-сжатые файлы

Linux-Команды для tar архивов ".tar"/".tar.xz"/".tar.gz"(or ".tgz")/".tar.bz2":
Options/flags:

- option:

`-C` это типа "текущая (**C**urrent) директория" для команды tar или еще можно сказать так: "директория of текущего контекста (of Current **C**ontext) для команды tar или еще можно сказать так: "абсолютный путь директории" - относительно которой можно будет еще указать дополнительный аргумент другую (относительную) директорию (с относительным путем) - которая будет упаковываться в архив или в которую будет распаковываться архив (p.s. если операция создания архива то тогда "дополнительная" директория является обязательной и должна указываться (если это та же самая директория то указать в качестве дополнительной директории точку), а для операции extract archive можно Не указывать "дополнительную" директорию - тогда будет просто использоваться директория указанная в опции -C) (p.s. эту директорию нужно будет предварительно создать если она не существует - иначе будет error во время выполнения команды)
(p.s. на самом деле можно типа Не указывать опцию -C, А указывать только дополнительную директорию - НО тогда будет Очень Некорректное поведение команды: а именно: будет затрагиваться начальный префиксный абсолютный путь при архивации, то есть тогда внутри of распакованного архива будет такой полностью скопированный длинный файл: /home/testusr/TestDir/d2/f2 Вместо такого ожидаемого: d2/f2)

- option:

-c означает create (tar archive)

- option:

-v означает вывод логирования (в консоль терминала) во время выполнения команды

- option:

-f означает что после списка опций указывается путь для target tar файла (то есть тот файл который будет создан - если это команда для создания архива (а не для распаковки)

- option:

-t означает list contents of tar archive (вывод содержимого of архива в консоль терминала)

- option:

-x означает "extract" (распаковать)

- option:

-z это опция которую нужно добавлять если операции для архива с типом gzip

- option:

-j это опция которую нужно добавлять если операции для архива с типом bzip2

- option:

p.s. остальные options если что читать в man-е

p.s. Для типа архива "tar.xz" возможно нужно еще предварительно установить tool:
`sudo apt install xz-utils`
(а далее все команды для ".tar.xz" точно такие же как для типа ".tar")

Example:
create new tar archive from dir `~/TestDir`:

```
testusr@some-pc:~$ tar -vcf ~/Data/TestTar.tar -C ~/TestDir .
./
./d2/
./d2/f2
./d1/
./d1/sd1/
./d1/sd1/f1
./d1/sd2/
```

(p.s. в конце команды - точка - это как раз (обязательный)-"дополнительный относительный" путь)
кстати: про итоговые размеры архивов:

```
testusr@some-pc:~$ ls -lsh ~/Data | grep .tar
12K -rw-rw-r-- 1 testusr testusr  10K Nov 30 20:06 TestTar.tar
4.0K -rw-rw-r-- 1 testusr testusr  261 Nov 30 23:41 TestTar.tar.bz2
```

```
4.0K -rw-rw-r-- 1 testusr testusr  256 Nov 30 23:41 TestTar.tar.gz
```

Example:

show contents of tar archive:

```
testusr@some-pc:~$ tar -tf ~/Data/TestTar.tar
./
./d2/
./d2/f2
./d1/
./d1/sd1/
./d1/sd1/f1
./d1/sd2/
```

p.s. а если добавить опцию `-v` то будет отображаться Примерно также как `ls -l` (то есть еще будут отображаться permissions, creation date (кстати дата это: когда файл был создан, А Не когда файл "попал" в архив), т.д. и еще будут даже отображаться: file sizes (для файлов которые внутри of tar архива))

Example:

А если **открыть** этот tar файл **в vs code-е в HexEditor-е**

то там отображается примерно так:

- список of nested dirs and files (тоже самое - как выводится в команде: `$ tar -tf ~/Data/TestTar.tar`) (p.s. то есть примерно тоже самое как если бы написали ls -R для исходной директории - которую мы потом упаковывали в tar)
- после каждой строки (то есть после каждого пути sub-директории/файла) отображается:

```
<какие-то "мета" данные: типа "тип" of текущего элемента:
директория/файл, и т.д., и затем еще набор нулей (пустые символы)>
```

- и если "текущий элемент" это "файл" - то тогда ниже еще будет отображаться контент этого файла

```
(example): hello_1
```

А если это Не просто tar архив, А tar.gz или tar.bz2 то тогда это будет очень компактный архивный файл в котором Все символы будут кракозябрами

Example:

check content of archive:

```
$ tar -tf ~/Downloads/jre-8u441-linux-x64.tar.gz
```

p.s. output

jre1.8.0_441/<... т.д. ...>

jre1.8.0_441/<... т.д. ...>/<... т.д. ...>

p.s. то есть внутри архива находится директория "jre1.8.0_441"

Example:

extract tar archive to some directory:

```
tar -xf ~/Data/TestTar.tar -C ~/Data/TestTar
```

p.s. more real example (более реальный пример):

```
$ sudo tar -xzf ~/Downloads/go1.23.3.linux-amd64.tar.gz -C /usr/local
```

p.s. result: "`/usr/local/go`"

p.s.

`-f` это source tar archive - который будет распаковываться

p.s. если добавить опцию -v то в output-е в консоли терминала будет отображаться также как в команде `$ tar -tf ~/Data/TestTar.tar`)

`Environment Variable` in Ubuntu:

"environment variable" это переменная окружения

("окружение" это типа весь текущий хост с текущей операционной системой, то есть это значит что такая переменная будет задана на "уровне системы" и будет "видна" **во всех терминалах**)

Name conventions for env variables:

если несколько частей в переменной то в качестве "типа слитного разделителя" использовать "нижнее подчеркивание" ("_"):

examples:

- LC_NUMERIC
- BASH_DIR
- GO_1_23_4
  (типа это конкретная версия of goLang-a (типа означает версию "1.23.4"))
- т.д.

p.s. очень **НЕ рекомендуется** использовать **другие разделители** (точки, пробелы, т.д.) так как иначе потом при выполнении операции могут быть ошибки/баги (так как именно эти другие "сложные" разделители очень специфично (по другому более сложному синтаксису а то и даже хардкодом) нужно обрабатывать в скриптах) (причем переменная может быть не для own программы, а для сторонней программы, поэтому тем более обработка будет например чужими external скриптами, так что тем более Не назначать "заведомо возможно-багованные" названия для env variable)

command for: show concrete environment variable by name:

```
testusr@some-pc:~/Data$ echo $someEnv1
some value 1
```

command for: show all environment variables:

```
testusr@some-pc:~/Data$ env
<... other environment variables ...>
someEnv1=some value 1
<... other environment variables ...>
```

grep by environment variable name:

```
testusr@some-pc:~/Data$ env | grep someEnv1
someEnv1=some value 1
```

How to create "not-persistent" (local, temp) env variable (means: only for concrete terminal-tab):

- Option:

```
someuser@somepc:~/somepath$ export FOO=abc
```

- OR Option:

создать файл
например
файл config.env:

```
FOO=abc
```

// далее:

```
someuser@somepc:~/somepath$ source config.env
```

How to show env:

- Option:

```
someuser@somepc:~/somepath$ echo $FOO
```

p.s. output

```
abc
```

- OR Option:

```
someuser@somepc:~/somepath$ env
```

How to create "persistent" env variable:

нужно внести изменения в специальный файл:
Здесь 2 варианта:

- Option:
  переменная окружения только **для одного конкретного юзера** of компьютера
  - тогда:

тогда это файл: ~/.bashrc

(p.s. иногда в некоторых документациях пишут (без всяких пояснений) другое название: ~/.bash_profile - это оказывается просто такой файл используется в операционной системе Apple (Macbook, т.д.) (А Не в Linux-е) (а для Linux-а это файл: ~/.bashrc))

- затем:

открыть этот файл в каком-нибудь редакторе (nano/vscode/т.д.)
и написать в конце файла:

```
export someEnv1="some value 1"
```

(p.s. указывая слово export)
и затем сохранить файл.
p.s. если несколько переменных, то разделять символом Enter (ну то есть писать каждую переменную на отдельной строке)
example:

```
export someEnv1="some value 1"
export anotherEnv1="another value 1"
```

- затем:

затем нужно рестартануть терминал (то есть: либо рестартануть терминал instance, либо переоткрыть terminal-tab, либо открыть отдельно новую вкладку терминала (new terminal-tab))
тогда:
в этом "переоткрытом" терминале подтянутся изменения of переменных окружения

- затем:

затем можно проверить что переменная выставилась:
(причем тОлько для текущего юзера, (а для других юзеров эта переменная Не выставилась)
затем залогиниться под текущим юзером:

```
$ sudo testusr
```

затем:

```
$ echo $someEnv1
```

p.s. отобразилось выставленное значение
затем проверить что для других пользователей переменная Не выставилась:
то есть залогиниться под каким-нибудь другим юзером (например под юзером root)

```
$ sudo su
```

затем:

```
$ echo $someEnv1
```

p.s. ничего не отобразилось (или если это существующая переменная окружения то тогда отобразилось старое Неизмененное значение)

- или: переменная окружения **для всех юзеров** этого компьютера
  - тогда:

тогда это файл: /etc/environment

- затем:

открыть этот файл в каком-нибудь редакторе (nano/vscode/т.д.)
и написать в конце файла:

```
someEnv1="some value 1"
```

(p.s. Не указывая слово export)
p.s. если несколько переменных, то разделять символом Enter (ну то есть писать каждую переменную на отдельной строке)
example:

```
someEnv1="some value 1"
anotherEnv1="another value 1"
```

p.s. и сохранить файл

- затем:

затем можно проверить "тестово" - До рестарта компьютера
(типа проверить что всё правильно по синтакису написали и что значения ожидаемые)
таким образом:
в терминале прописать:

```
$ source /etc/environment
$ echo $someEnv1
$ echo $anotherEnv1
```

- затем:

затем нужно рестартануть компьютер (то есть Не просто терминал а именно компьютер (p.s. это логично-понятно так как переменная была задана для всех юзеров компьютера))
тогда:
подтянутся изменения of переменных окружения

- затем:

затем можно проверить что переменная выставилась:
(причем для всех юзеров)
для этого:
например сначала залогиниться под текущим юзером:

```
$ sudo testusr
```

затем:

```
$ echo $someEnv1
```

p.s. отобразилось выставленное значение
затем проверить для какого-нибудь другого юзера
(то есть залогиниться под каким-нибудь другим юзером (например под юзером root))

```
$ sudo su
```

затем:

```
$ echo $someEnv1
```

p.s. тоже отобразилось выставленное значение

p.s. Если переменная уже определена в этом файле (в ~/bashrc or в /etc/environment) Но например нужно изменить значение:
то можно:

- либо прямо изменить контент-значение в уже существующей строке где определена эта переменная
- либо:
  ниже (на отдельной новой строке, причем неважно нА-сколько строк ниже, главное что ниже, так как более нижние строки "override-дят" более верхние строки)
  написать новое определение для этой переменной
  p.s. а если нужно добавить подстроку к уже существующему старому значению то использовать стандартный bash синтаксис для получения значения переменной: то есть выражение $someEnv это существующее значение of переменной someEnv
  example:
  типа старая существущая строка:

```
someEnv="/usr/local"
```

  а например ниже прописали так:

```
someEnv="$someEnv/bin"
```

  то есть в итоге самое "свежее" значение будет равно: /usr/local/bin

Update:
в etc/environment Не проддерживается функционал: указывания существующего значения через $
например нельзя ввести
someVar="value"
someVar="${someVar}/additional" (p.s. пробовала разные варианты синтаксисов: и без кавычек и без фигурных скобок и т.д. - ничего Не разрешается)
(вероятно так как это очень простой конфиг в котором типа Не поддерживается сложный синтаксис типа указывания of существующего значения переменной через амперсанд $) (так что в этом файле получается тОлько редактировать текстом существующий контент в существующей строке of существующей переменной

example: как изменить значение of системной переменной PATH:

например добавить еще один путь "/usr/local/go/bin" в существующую переменную окружения PATH
(p.s. например пишем в файле ~/.bashrc (или в терминале) (или в каком-нибудь скрипте))

```
export PATH="$PATH:/usr/local/go/bin"
```

p.s. здесь написано что типа конкатенация:
<env PATH> = <existing old value of env PATH> + <appended new sub-value>
(<existing old value of env PATH> = `$PATH`)
(<appended new sub-value> = `:/usr/local/go/bin`)
p.s. в переменной PATH пути "разделяются" символом двоеточие ":", поэтому перед "новым-дополнительным" значением-путём добавлен этот "разделитель" (`:`)
p.s. затем проверим новое значение of этой переменной PATH
example:

```
testusr@some-pc:~/Data$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin:/snap/bin:/usr/local/go/bin
```

---

history of bash-terminal commands:

```
history
```

Дописать, выводит **Не всю** историю!

---

ubuntuOS-package-manager (programs-manager) `apt` and `dpkg`:

apt - это менеджер программ (пакетов/приложений) для Debian-based операционных систем
(in english: package management system)

`$ sudo apt **update**`
это обновление списка репозиториев откуда скачиваются пакеты
(то есть раньше в конфиге где прописаны списки репозиторией были указаны ссылки на более старые репозитории, а после выполнения команды стали в конфиге обновились ссылки: стали ссылки на более новые репозитории)

`$ sudo apt **upgrade**`

это непосредственно - обновление текущих (уже установленных) пакетов (from current более старой версии на более новые версии)
(то есть в соответсвии с более новыми ссылками репозиториев: для установленных програм скачаются самые свежие версии (latest versions), ну и не только скачаются а еще установятся эти latest версии (то есть установленные программы обновятся))

`$ apt list --upgradable`
это output: какие ссылки на какие репозитории обновились

How to install package (program):
p.s. если установка будет через утилиту apt и если установка именно по названию программы (то есть будет неявное скачивание "установочного файла" из "configured" source-репозиториев), то тогда рекомендуется До установки выполнить команду "`$ sudo apt update`"
p.s. <DebFilePath> это именно full absolute path of .deb файла (то есть например если файл в текущей-навигированной директории то тогда full path через точку: ./some_file.deb)
(так как иначе apt Не видит этот .deb файл (если снавигироваться(cd) например в Downloads) (вероятно там есть что-то похожее как в утилите tar (типа может там тоже можно указать какой-то параметр-аргумент который будет задавать директорию в которой находится .deb файл)))
then:

- Option 1:
  Через пакетный-менеджер "apt":
  `$ sudo apt install <program_name>` или `$ sudo apt install <DebFilePath>`
- OR: Option 2:
  Через утилиту "dpkg":
  `$ sudo dpkg -i <DebFilePath>`
  затем выполнить команду:
  `$ sudo apt --fix-broken install` или более сокращенно: `$ sudo apt -f install`
  (p.s. после этого Не нужно повторно выполнять dpkg -i)
  эта команда означает:
  установить зависимости (зависимые пакеты для этой устанавливаемой программы (то есть: not installed/not found program dependencies)) - так как команда dpkg Не устанавливает зависимости
  (p.s. example error text (after command: `dpkg -i` (but before command: `apt -f install`)): "<program x> depends on libminizip1; however: package libminizip1 is not installed.")

p.s. чтобы получить подсказку по синтаксису команды (`apt --fix-broken install`)
можно выполнить:
`$ sudo apt install`
p.s. without arguments
p.s. output contains:
"Unmet dependencies. Try '**apt --fix-broken install**' with no packages"

p.s.
после установки .deb пакета - этот файл .deb можно удалить
(`$ rm <DebFilePath>`)

p.s.
"dpkg -i" VS "apt install"
https://unix.stackexchange.com/questions/159094/how-to-install-a-deb-file-by-dpkg-i-or-by-apt
краткий пересказ:
(программа dpkg это более низкоуровневая программа чем apt)
(a "apt" internally uses и calls (вызывает) программу dpkg, а потом еще неявно устанавливает зависимые пакеты,
то есть "`$ apt install`" это примерно равно "`$ dpkg -i; apt -f install;`")

How to show installed programs которые были установлены утилитой "apt":
```
sudo apt list --installed
```

How to uninstall program которая была установлена утилитой "apt":
- option: delete only program (not delete configs of that program)

```
sudo apt remove <programName>
```
- option: delete program and configs of that program

```
sudo apt-get --purge remove <programName>
```
example:
```
sudo apt-get --purge remove dolphin
```

How to show installed programs которые были установлены утилитой "dpkg":
```
dpkg --list
```
example:
```
$ dpkg --list | grep -i teamv
```
example output:
```
ii  teamviewer-host                                15.60.3
```

How to uninstall program которая была установлена утилитой "dpkg":

- option: delete only program (not delete configs of that program)

```
sudo dpkg -r <programName>
```

- option: delete program and configs of that program

```
sudo dpkg --purge <programName>
```

example:

```
sudo dpkg --purge teamviewer-host
```

Symbolic Link (symlink) tutorial:

how to create symlink:
example:

```
sudo ln -s /etc/someDir/someFile.bin /usr/bin/someExecFile
```

p.s. то есть: файл "`/usr/bin/someExecFile`" = файл "`/etc/someDir/someFile.bin`"

how to show symlinks:

- install tool:

```
sudo apt install symlinks
```

- then:
    - option:

show symlinks которые существуют на эту директорию (или на вложенные директории Но именно тОлько первого уровня вложенности)

```
$ symlinks -v <dirPath>
```

example:

```
$ symlinks -v /usr/bin
```

example output:

```
< ... и т.д. ... >
absolute: /usr/bin/someExecFile -> /etc/someDir/someFile.bin
< ... и т.д. ... >
```

example:

```
symlinks -v /
```

example output:

```
relative: /lib64 -> usr/lib64
relative: /lib -> usr/lib
relative: /bin -> usr/bin
relative: /sbin -> usr/sbin
```

    - or option:

show symlinks которые существуют на эту директорию (или на вложенные директории Всех вложенных уровней)

```
$ symlinks -v <dirPath>/*
```

example:

```
$ symlinks -v /*
```

example output:

```
< ... и т.д. ... >
absolute: /bin/vim -> /etc/alternatives/vim
< ... и т.д. ... >
```
