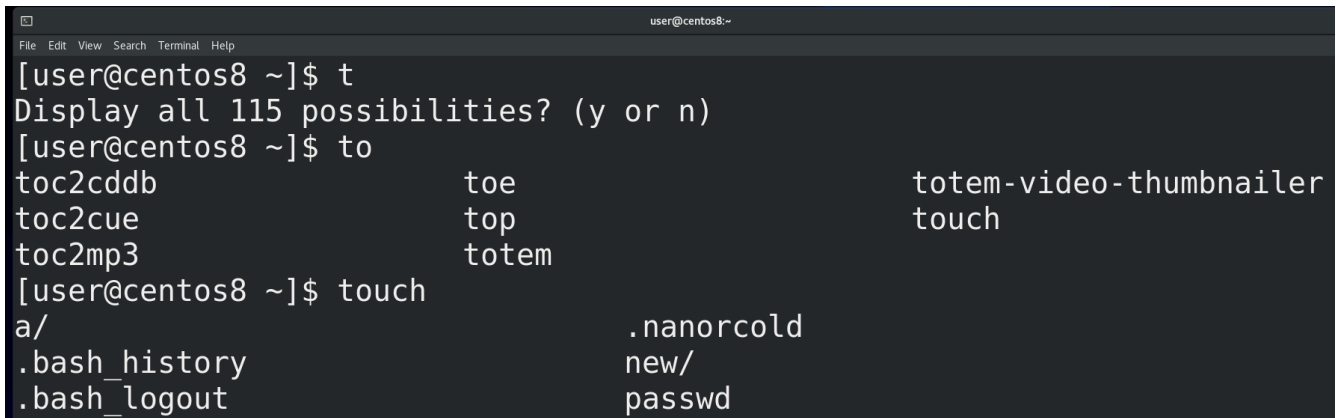
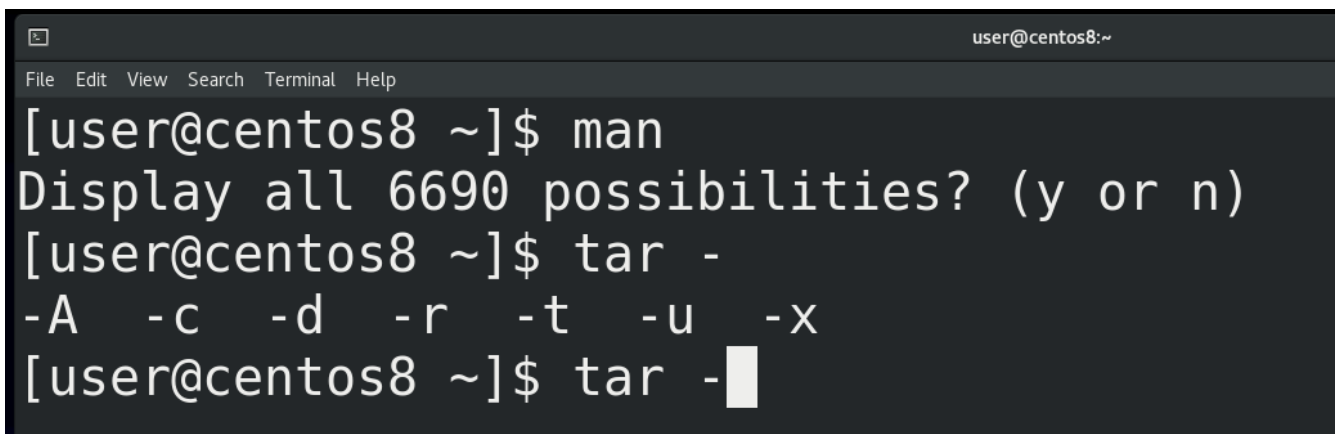


В теме “Текстовый интерфейс пользователя” я рассказал про интерпретатор командной строки bash. Давайте разберём, что же он умеет.



```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ t  
Display all 115 possibilities? (y or n)  
[user@centos8 ~]$ to  
toc2cddb          toe          totem-video-thumbnailer  
toc2cue           top          touch  
toc2mp3           totem  
[user@centos8 ~]$ touch  
a/                .nanorcold  
.bash_history     new/  
.bash_logout     passwd
```

Тогда я упомянул, что bash умеет дополнять команды – допустим, вы пишете часть команды ( to, tou, touch file), нажимаете tab и bash дописывает команду или файл за вас. Если вариантов несколько – нажимаете tab два раза и видите все варианты. Но есть два типа дополнения – простое и продвинутое. В случае простого дополнения у вас просто дописывается сама команда или файл. Продвинутое дополнение может добавлять опции и значения.



```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ man  
Display all 6690 possibilities? (y or n)  
[user@centos8 ~]$ tar -  
-A  -c  -d  -r  -t  -u  -x  
[user@centos8 ~]$ tar -█
```

Мы с вами на виртуалке поставили CentOS с графическим интерфейсом, то есть user-friendly версию, поэтому у нас сейчас стоит продвинутое дополнение. Из пройденных команд это можно увидеть по команде man – если написать man и нажать два раза tab, то bash скажет, что у нас больше 6 тысяч вариантов – всё это страницы документации, то есть, значения для man. Или, например, команда для архивации - tar – о ней мы поговорим в другой раз. Но, с точки зрения дополнения, если написать tar и нажать tab, то сперва появится дефис, а после двух нажатий tab появятся опции этой команды. В отличие от пройденных команд, где всякие ключи не обязательны, у tar ключи обязательны, поэтому bash сразу предлагает их написать.

```
user@centos8:~$ sudo dnf -y remove bash-completion
[sudo] password for user:
Dependencies resolved.

=====
Package Arch [user@centos8 ~]$ man
=====a/ .nanorc
Removing: .bash_history .nanorcolo
bash-completion noarch .bash_logout new/
.bash_profile passwd
bashrc passwd-cv
```

Теперь давайте рассмотрим простой вариант дополнения. Обычно, когда стоит минимальная система без графического интерфейса, дополнение простое. Функционал продвинутого дополнения ставится с пакетом `bash-completion`. Давайте я удалю этот пакет и мы посмотрим что будет. Для этого я пишу `sudo dnf -y remove bash-completion` и ввожу пароль моего пользователя. В рамках запущенной `bash` сессии ничего не изменится, но если открыть новое окно терминала – то тут у нас дополнение будет работать в урезанном виде. Например, пишу `man`, нажимаю `tab` – а он мне предлагает файлы вместо страниц документации. Или пишу `tar`, нажимаю `tab` – опять же файлы. Сами команды и файлы будут дописываться – если написать `to` и два раза нажать `tab`, то мы увидим все команды, начинающиеся на эти буквы. Но сразу после команд при нажатии `tab` будут предлагаться файлы, и не важно, работает команда с файлами или нет. Поэтому установим обратно `bash-completion` – `sudo dnf -y install bash-completion`.

```
user@centos8:~$ alias
alias egrep='egrep --color=auto'
alias fgrep='fgrep --color=auto'
alias grep='grep --color=auto'
alias l.='ls -d .* --color=auto'
alias ll='ls -l --color=auto'
alias ls='ls --color=auto'
alias vi='vim'
```

Ещё в `bash` можно настроить алиасы. Алиасы дают возможность настроить какие-то свои команды на основе других. Если запустить команду `alias`, можно увидеть список заданных алиасов для текущей `bash` сессии.

Я говорю `bash` сессия, но что это такое? Когда вы запускаете эмулятор терминала, у вас запускается `bash` сессия. Если вы запустите другое окно – будет другая сессия. Напишите `bash` – ещё одна сессия. Для каждой сессии `bash` запускается “с нуля”, то есть он считывает файл настроек. Это значит, что если вы измените файл настроек, то в запущенном `bash`-е ничего не

изменится. Но если написать `bash` или запустить новое окно, то есть запустить новую сессию `bash`, то новая сессия эти изменения увидит. Сейчас, на примере алиасов, разберёмся.

```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ l.  
. .ICEauthority .vboxclient-display.pid  
.. .lessht .vboxclient-display-svga-x11.pid  
.bash_history .local .vboxclient-draganddrop.pid  
.bash_logout .mozilla .vboxclient-seamless.pid  
.bash_profile .nanorc .viminfo  
.bashrc .nanorcold .zcompdump  
.cache .passwd.swp .zshrc  
.config .pki  
.esd_auth .ssh  
[user@centos8 ~]$ alias ls  
alias ls='ls --color=auto'
```

Так вот, команда `alias` показала нам текущие алиасы. Тут мы видим пару знакомых команд, связанных с `ls` и `vi`. И так, здесь написано, что команда `l` точка (`l.`) - это на самом деле команда `ls -d .* --color=auto`. `ls` показывает список файлов и директорий в текущей директории, ключ `-d` покажет сами директории, а не их содержимое, `.*` - все файлы и директории, у которых в начале названия есть точка, то есть скрытые файлы и директории. И `--color=auto` – раскрасить вывод. Так вот, если написать `l.` - мы увидим все скрытые файлы и директории в текущей директории. Если вы обратили внимание, у нас `ls` в принципе всегда раскрашивает вывод – а это потому что есть алиас на сам `ls` с опцией `--color=auto`. Тут же есть алиас на `vi` – то есть когда мы запускаем `vi` у нас запускается `vim`.

```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ alias vi=nano  
[user@centos8 ~]$ alias show="tail -5 /etc/passwd"  
[user@centos8 ~]$ show  
b.allen:x:1021:1002::/home/j  
b.wayne1:x:1022:1002:Birthd  
b.wayne12:x:1023:1003:Birth  
b.wayne123:x:1024:100:Birth  
b.wayne1234:x:1025:1002:Birth  
[user@centos8 ~]$  
[user@centos8 ~]$ show  
bash: show: command not found...  
Failed to search for file: Failed to lo  
[user@centos8 ~]$
```

Давайте сами зададим какой-нибудь `alias`. Например, `alias vi=nano`. Теперь у нас при запуске `vi` будет открываться `nano` – `vi file`. Или, допустим, `alias show="tail -5 /etc/passwd"`. Обратите внимание, что я взял правую часть в кавычки, а в примере с `nano` я этого не делал. Сами подумайте, почему, если не знаете – попробуйте ввести без кавычек и проверить. Ну и если вам пока это сложно понять – спрашивайте в комментариях, я отвечу. Так вот, чуть ранее я говорил про `bash` сессии. И те алиасы, которые мы создали, сохранились только в текущей сессии. Это значит, что если запустить новое окно – то там наших алиасов уже не будет.

```
GNU nano 2.9.8 /home/user/.bashrc

1 ## .bashrc
2
3 # Source global definitions
4 if [ -f /etc/bashrc ]; then
5     . /etc/bashrc
6 fi
7
8 # User specific environment
9 PATH="$HOME/.local/bin:$HOME/bin:$PATH"
10
```

Если мы хотим, чтобы наши алиасы остались навсегда, мы должны их написать в файле настроек bash - ~/.bashrc, который находится у нас в домашней директории – nano ~/.bashrc . Как вы видите, этот файл не пустой, здесь есть какие-то настройки. Некоторые строчки начинаются со знака решётки (#). Такие строчки называются закомментированными – как правило, программы такие строчки не считывают, они больше нужны для людей.

```
GNU nano 2.9.8 /home/user/.bashrc

14 # User specific aliases and functions
15
16 # My custom aliases
17
18 alias show="grep user /etc/passwd"
19

^G Get Help
^X Exit
```

```
[user@centos8 ~]$ show
tss:x:59:59:Account used by the trousers package to sandbo
/null:/sbin/nologin
qemu:x:107:107:qemu user:/:/sbin/nologin
usbmuxd:x:113:113:usbmuxd user:/:/sbin/nologin
saslauth:x:991:76:Saslauthd user:/run/saslauthd:/sbin/nolo
```

Обычно без разницы, где добавлять новые строчки, но для удобства давайте спустимся в самый низ и добавим там. Напишем комментарий - # My custom aliases . Кстати, в nano можно нажать Esc+3 и nano сам добавит или уберёт знак решётки с текущей строки. Дальше я пишу свои алиасы, по одному на каждую строку – alias show="grep user /etc/passwd" и закрываю файл. В текущей баш сессии ничего не изменится, поэтому я запускаю новое окно и уже там эти алиасы видны и работают.

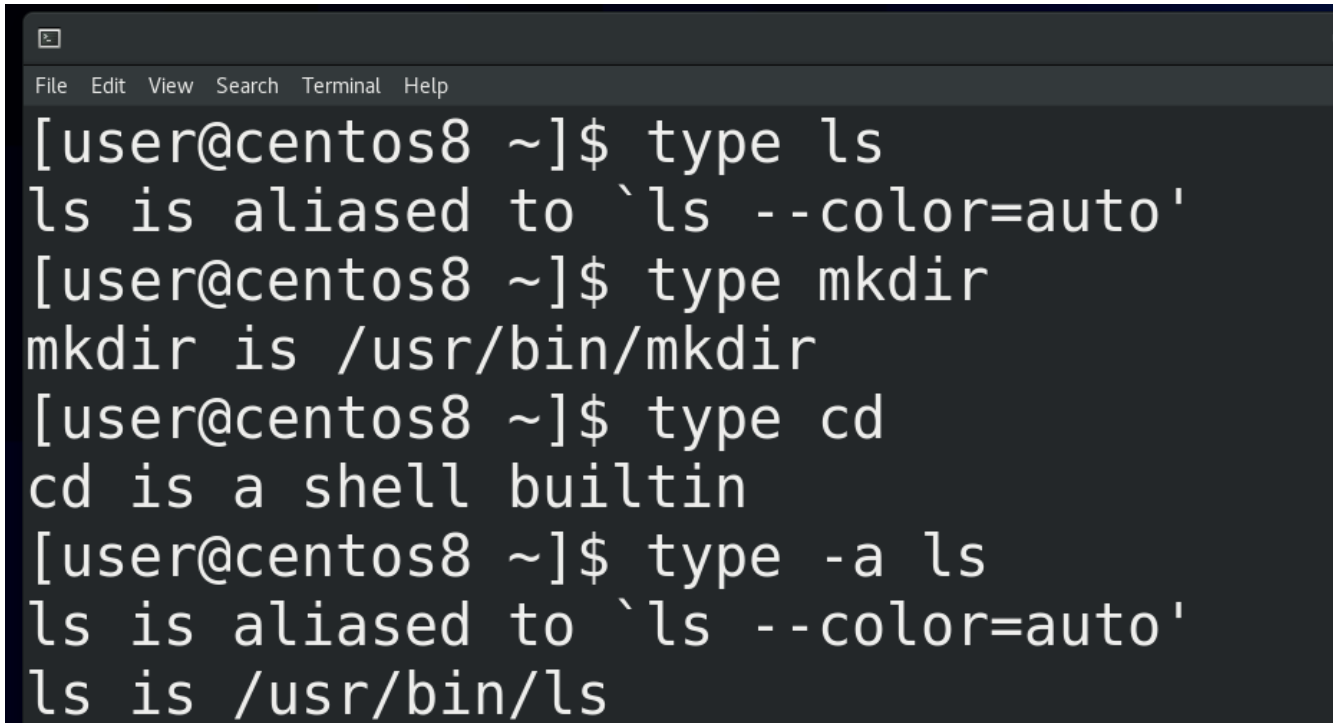
```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ ls  
a Documents FILE Music Pictures Templates Videos  
Desktop Downloads FILE new Public test  
directory file is passwd temp this  
[user@centos8 ~]$ \ls  
a Documents FILE Music Pictures Templates Videos  
Desktop Downloads FILE new Public test  
directory file is passwd temp this  
[user@centos8 ~]$ "ls"  
a Documents FILE Music Pictures Templates Videos  
Desktop Downloads FILE new Public test  
directory file is passwd temp this  
[user@centos8 ~]$ command ls  
a Documents FILE Music Pictures Templates Videos  
Desktop Downloads FILE new Public test  
directory file is passwd temp this  
[user@centos8 ~]$
```

Алиасы нужны пользователям для упрощения работы, бывают сложные и длинные команды, которые вы часто должны вбивать. И чтобы облегчить себе жизнь, вы можете создать для себя алиасы. Но бывают случаи, когда вам нужен не алиас, а сама команда. Допустим, в системе есть алиас вместо `vi` запускать `vim`, или окрашивать вывод `ls`. А что, если мне нужно разок запустить обычный `ls`? Есть несколько способов это сделать – поставить обратный слэш перед командой - `\ls`, либо взять команду в кавычки – `"ls"`, либо написать перед ней `command` - `command ls`.

```
user@centos8:~  
File Edit View Search Terminal Help  
[user@centos8 ~]$ alias  
alias egrep='egrep --color=auto'  
alias fgrep='fgrep --color=auto'  
alias grep='grep --color=auto'  
alias l.='ls -d .* --color=auto'  
alias ll='ls -l --color=auto'  
alias ls='ls --color=auto'  
alias vi='vim'
```

Кстати, как вы, возможно, заметили, список алиасов от команды `alias` не малый, но в файле `~/.bashrc` их не было. Файл `.bashrc` распространяется только на моего пользователя, потому что он в моей домашней директории. Но есть файл `/etc/bashrc`, который распространяется

на всех пользователей в системе. Допустим, если вы хотите создать alias, который бы работал у всех пользователей – создаёте его именно в этом файле. Но, если полистать этот файл, вы всё равно не найдёте те же алиасы на ls. Поэтому ещё одна задача для вас – постарайтесь найти файл, в котором указаны алиасы, которые у нас есть по умолчанию. Всё что нужно чтобы найти – мы уже проходили.

A screenshot of a terminal window with a dark background and light-colored text. The window has a menu bar at the top with 'File', 'Edit', 'View', 'Search', 'Terminal', and 'Help'. The terminal shows the following commands and their outputs:

```
[user@centos8 ~]$ type ls
ls is aliased to `ls --color=auto'
[user@centos8 ~]$ type mkdir
mkdir is /usr/bin/mkdir
[user@centos8 ~]$ type cd
cd is a shell builtin
[user@centos8 ~]$ type -a ls
ls is aliased to `ls --color=auto'
ls is /usr/bin/ls
```

Мы разобрались, что bash может запускать как команды, так и алиасы. Сами команды можно разделить на 2 типа – внешние и внутренние. Какие-то команды встроены в bash, они называются внутренними – например, тот же cd. Но большинство команд лежат на файловой системе в специальных директориях и называются внешними. Чтобы как-то отличать, что это за команда – внутренняя, внешняя или алиас – есть команда type. Допустим, type ls, type mkdir, type cd. В случае ls – это алиас, в случае mkdir bash показывает путь к программе, то есть это внешняя программа, а в случае с cd bash говорит, что это встроенная в оболочку команда. Но ведь ls – это алиас сам на себя, как понять, что там стоит за этим алиасом? Для этого можно использовать type с опцией -a – type -a ls. Теперь мы видим, что ls – это /usr/bin/ls – то есть внешняя программа.

И так, мы узнали, что у нашего интерпретатора командной строки - bash – есть дополнения – простые и продвинутые. Также мы поговорили про алиасы, внутренние и внешние программы. Но bash может делать гораздо больше, поэтому будет несколько тем, посвящённых bash.