Лабораторная работа № 2.
Работа с терминалом и оболочкой Bash

Введение

В этой лабораторной работе мы узнаем, чем терминал отличается от консоли, как работает оболочка командной строки, каким сочетанием клавиш можно удивить даже бывалых сисадминов.

Графический интерфейс в современных системах

В современных ОС графический интерфейс нашел широкое применение – это наглядно и удобно. Однако для выполнения задач администрирования одного только графического интерфейса, как правило, бывает недостаточно, особенно в задачах автоматизации. Даже в ОС Windows, для которой графический интерфейс управления создавался как основное средство взаимодействия с пользователем, все более активно используются CMD и PowerShell, а некоторые настройки, например, для Microsoft Exchange, можно сделать только из командной строки.

Аппаратный терминал

На заре зарождения вычислительной техники компьютеры позаимствовали у телеграфа так называемые телетайпы (teletype, TeleTYpewriter или сокращённо TTY) — обычные электромеханические печатные машинки, которые использовались для передачи текстовых сообщений между двумя абонентами. Однако использование таких инструментов ввода требовало предельной осторожности в наборе команд, поэтому очень скоро разработчики реализовали технологию редактирования вводимых команд, которую назвали «Line Discipline».

Термин Line Discipline обычно переводят как «Дисциплина линии», что крайне неудачно. Слово «line» в этом контексте означает «строку», а «discipline» можно перевести как «правила», то есть более уместно говорить о «правилах обработки ввода строки».

Суть дисциплины линии очень проста: на стороне пользователя есть печатная машинка, а на стороне компьютера есть буфер, куда поступают напечатанные символы, и установлены правила, в соответствии с которыми содержание этого буфера может быть скорректировано при поступлении специальных символов.

Телетайп модель 33 ASR[3]

В 70-х годах телетайпы оснастили сначала принтерами, а затем дисплеями и научили получать обратную связь от компьютера, что позволило существенно расширить возможности дисциплины линии, например, стало возможным перемещать курсор по строке текста. Устройства, поддерживающие такой режим работы, стали называть терминалами (с англ. terminal или terminal end — оконечное устройство).

Много биткоинов утекло с тех времен, но базовые механизмы ввода до сих пор работают на тех же проверенных принципах. Поэтому при подключении к серверу через SSH ваш терминал передает по сети нажатие каждой клавиши, которое обрабатывается дисциплиной линии на стороне сервера, за счет чего работает, например, механизм автоматического дополнения команд по нажатию клавиши `Tab`.

Терминал VT100[3]

Системная и виртуальные консоли

Интерфейс для взаимодействия с пользователем в Linux называют консолью. Сразу после включения компьютера в консоли отображается ход загрузки, а после инициализации ядра консоль начинает предоставлять доступ к нескольким виртуальным консолям.

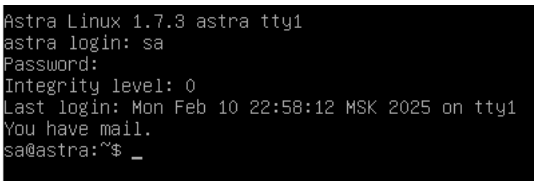
Для переключения между которыми используются горячие клавиши `Ctrl + Alt + F<N>`, где <N> – целое число от 1 до 8, например: `Ctrl + Alt + F1`. Виртуальные консоли позволяют выполнять несколько задач параллельно и мониторить работу системы.

В Astra Linux по умолчанию инициализируется 7 виртуальных консолей: tty1 … tty7. Консоли с первой по шестую являются текстовыми, а в седьмой консоли запускается графика. Каждой виртуальной консоли в системе соответствует одноименный символьный файл этого устройства в каталоге `/dev`. Например, консоли tty1 соответствует файл `/dev/tty1`. Устройство `/dev/console` автоматически привязывается к активной виртуальной консоли. Количеством виртуальных консолей в системе можно управлять (максимум 63 штуки) – для этого есть параметр NAutoVTs в конфигурационном файле `/etc/systemd/logind.conf`.

Примечание

Есть еще один способ, как можно быстро переключаться между виртуальными консолями. Сочетание клавиш `Alt + <стрелка влево>` или `Alt + <стрелка вправо>` позволяет переключаться между консолями по порядку. Однако после перехода в графику это сочетание перестает работать, и, чтобы перейти обратно в текстовую консоль, необходимо снова использовать `Ctrl + Alt + <F1-F6>`.

Для работы виртуальных консолей в системе Linux запускается несколько экземпляров приложения agetty, каждое из которых подключается к своему tty и реализует приглашение для ввода логина. После того как пользователь введет имя учетной записи, agetty запустит программу login и передаст ей введенное значение. Приложение login запросит у пользователя пароль, выполнит проверку учетных данных через PAM-стек и в случае успешной аутентификации запустит оболочку командной строки Bash из-под этого пользователя.

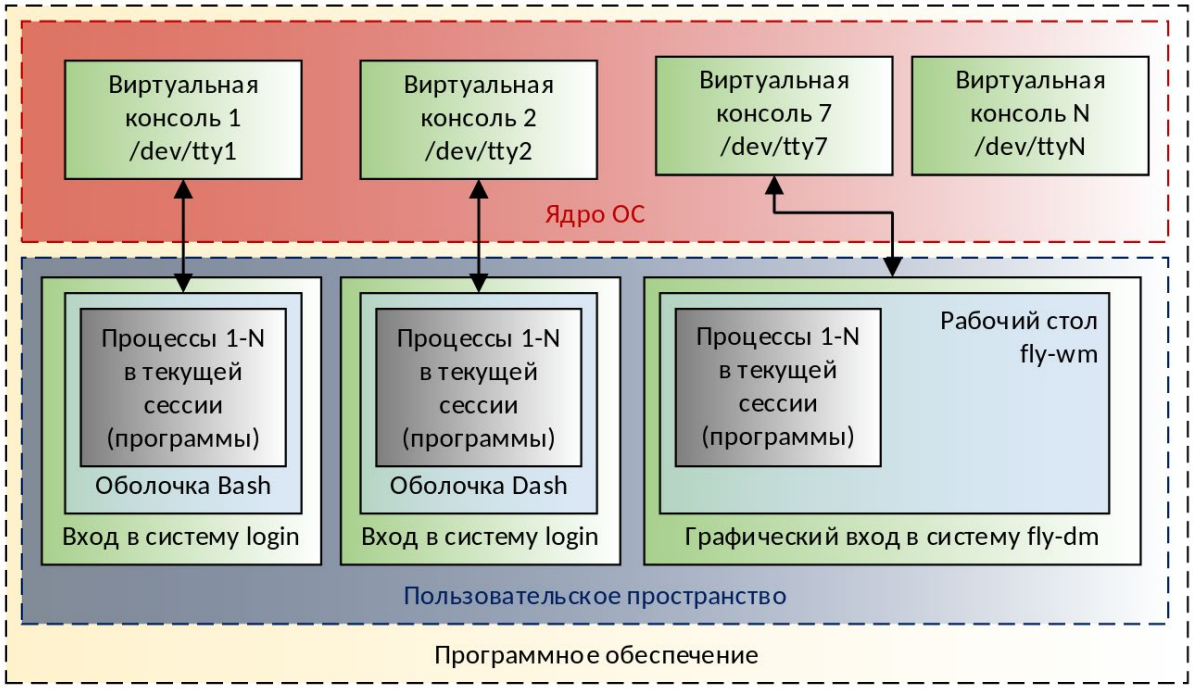


Оболочка (англ. shell) или интерпретатор командной строки — это системное приложение, главное предназначение которого состоит в том, чтобы предоставить пользователю возможность запускать другие приложения.

Существует большое количество различных оболочек, самой популярной из которых является Bash (в Astra Linux используется по умолчанию). Другие примеры: Dash (легковесный Shell, доступен, если запустить бинарник по адресу `/bin/sh`), Zsh. По синтаксису все оболочки довольно схожи, так что с первого взгляда не всегда понятно, какая именно оболочка запущена.

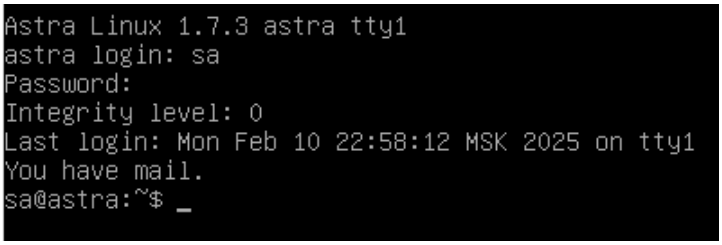
Важно

Не путайте понятия виртуальной консоли (или просто консоли) и оболочки. Консоль реализуется функциями ядра, а оболочка является обычной программой пользовательского пространства. На следующей схеме представлена логика взаимодействия между виртуальными консолями, утилитами входа в систему, оболочками и прикладным ПО (допущены некоторые упрощения).



Немного практики:

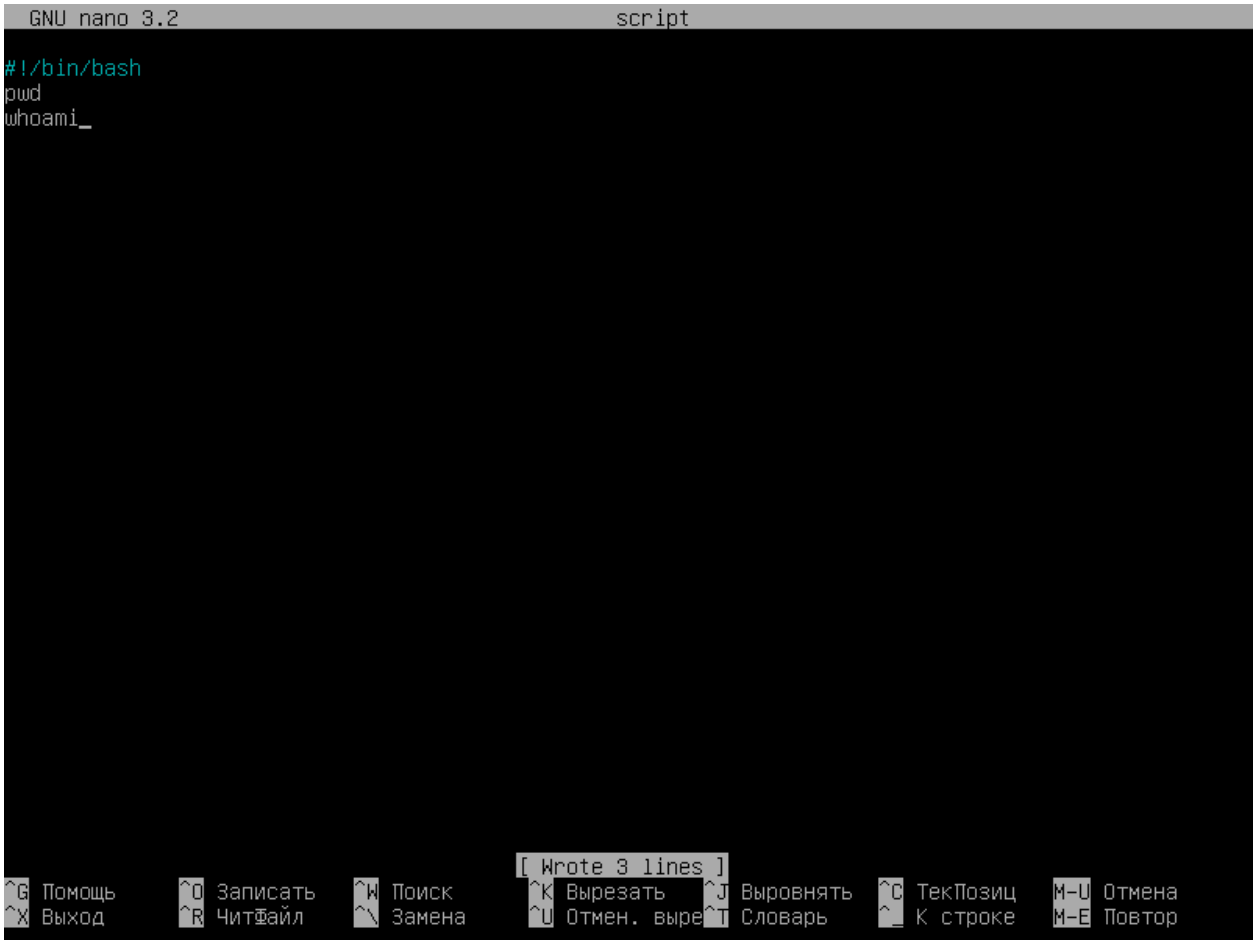
Перейдем в виртуальную консоль tty1 горячим сочетанием клавиш `Ctrl + Alt + F1` и введем учетные данные пользователя, который был создан при установке системы (далее под «админом» мы будем иметь в виду именно эту учетную запись).



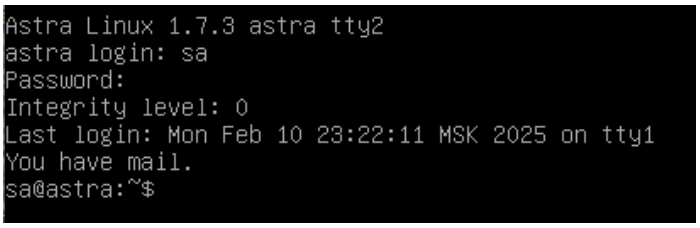
Выполним команду `nano script`, которая откроет нам текстовый редактор nano. Введем в редактор **текст**:

```
#!/bin/bash
pwd
whoami
```

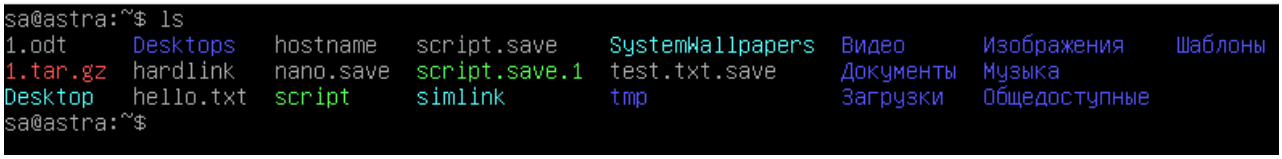
Сохраним файл нажатием `Ctrl + O` и `Enter`.



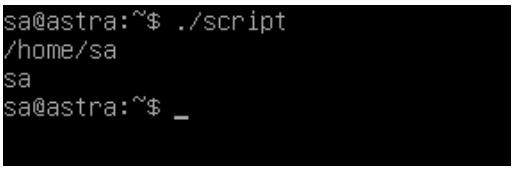
Перейдем в виртуальную консоль tty2 нажатием клавиш `Ctrl + Alt + F2` и выполним вход от имени админа



Посмотрим, какие файлы и каталоги у нас находятся в текущей директории с помощью команды `ls`.



Как видим, у нас появился файл script. Попробуем запустить его, выполнив команду `./script`, но… у нас это не получится. Система скажет нам, что у нас нет прав. Чтобы всё-таки запустить скрипт, выполним команду `sudo chmod u+x script`, которая разрешит выполнять этот файл, и команду `./script`, которая непосредственно запустит этот скрипт.



Скрипт успешно отработал, и на экран будут выведены два сообщения:

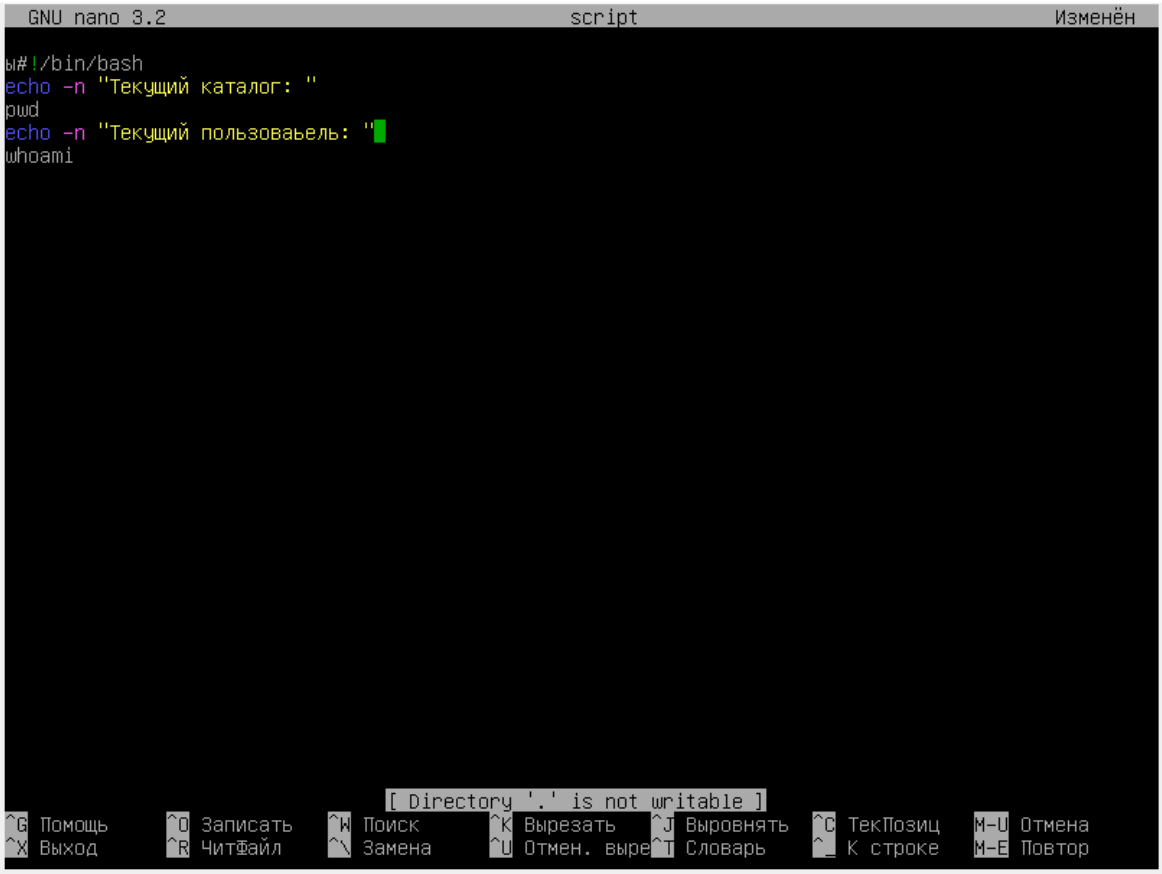
– /home/sa - текущая директория (в которой он был запущен)

– sa - и имя учетной записи (от которой был запущен скрипт).

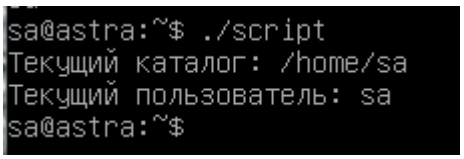
Переключимся на первую консоль tty1, использовав сочетание `Alt + <стрелка влево>`, и немного изменим наш скрипт:

```
#!/bin/bash
echo -n “Текущий каталог: ”
pwd
echo –n “Текущий пользователь: ”
whoami
```

Сохраним файл нажатием `Ctrl + O` и `Enter`.



Переключимся на вторую консоль tty2, использовав сочетание `Alt + <стрелка вправо>`, и запустим скрипт еще раз, выполнив команду `./script`:



Как видите, вывод скрипта ожидаемо изменился. Таким образом, переключаясь между консолями, можно эффективно работать на компьютерах с Linux даже без графической оболочки.

Примечание

Пользовательский процесс может изменять работу любого TTY-устройства, управляя соответствующим файлом в папке `/dev`. По сути, символьные файлы tty[1-7] не являются файлами с данными, хранимыми на диске, — это специальные файлы устройств, которые являются указателями на их драйверы.

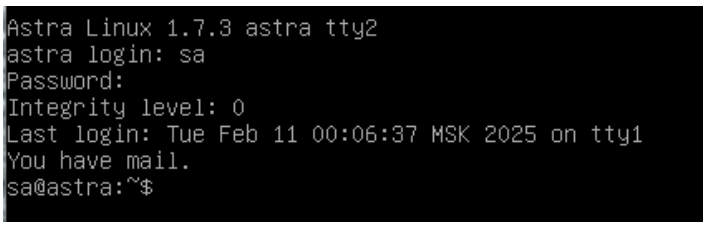
Поэтому, когда процесс обращается к файлу tty[1-7], он, по сути, работает с драйвером этого устройства. Естественно, для этого процесс должен обладать правами записи в этот файл. Когда пользователь входит в систему и подключается к определённому TTY, этот пользователь должен стать владельцем файла, соответствующего этому TTY.

Меняет права доступа к файлу терминала программа login, которая в свою очередь обладает необходимыми привилегиями, т.к. запускается от имени суперпользователя.

Перейдем в виртуальную консоль tty1 от имени админа.

Откроем текстовый редактор nano (просто введите nano и нажмите «Enter»).

Перейдем в виртуальную консоль tty2 и выполним вход от имени админа.



Выполним команду `ls -l /dev/tty[1-7]`, которая выведет на экран подробную информацию о файлах `/dev/tty1`, `/dev/tty2 … /dev/tty7`. Мы подробно разберем, что же означает этот вывод, в отдельных лабораторных этого курса, но сейчас нас интересует выделенный столбец. В нем содержится информация о владельце этого файла.

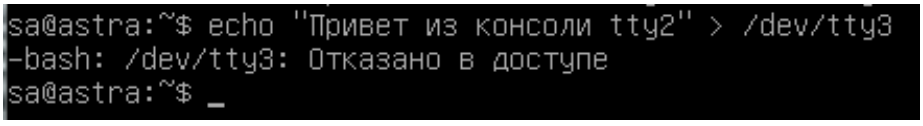
Как видите, владельцами первых двух виртуальных консолей является sa, с третьей по шестую – суперпользователь root, а владельцем седьмой консоли, на которой запущен графический интерфейс, является служебный пользователь fly-dm.

А теперь сделаем магический трюк. Оставаясь в консоли tty2, введем команду:

```
$ echo “Привет из консоли tty2” > /dev/tty1
```

Перейдем в виртуальную консоль tty1 увидим переданный нами текст.

Введенная нами команда перенаправила текст «Привет из консоли tty2» в консоль tty1 через файл консоли `/dev/tty1`. Однако, если мы опять перейдем на консоль tty2 и попытаемся отправить текст на консоль tty3, то получим сообщение о том, что у нас нет на это прав, потому что мы не являемся владельцем файла `/dev/tty3`.



Эмулятор терминала

Шли годы и в конце 90-х стали появляться первые графические оболочки для Linux. Для обеспечения обратной совместимости и возможности запускать консольные приложения из графической оболочки были разработаны специальные приложения – **эмуляторы терминала** (fly-term, xterm, ssh, telnet).

Предназначение у них ровно такое же, как и у виртуальных консолей, - доставлять ввод пользователя в запущенную программу и отображать вывод программы на дисплей. Но механизм работы в корне отличается, потому что сам эмулятор терминала работает в пользовательском пространстве, а для обратной совместимости работы приложений требовалось обеспечить работу механизмов на уровне ядра ОС. Для этого изобрели такое программное устройство, как **псевдотерминал** (*PTY, акроним Pseudo-TTY*). Сейчас, когда аппаратных терминалов нет, под «терминалом» понимают именно псевдотерминал.
Linux не ограничивает количество псевдотерминалов в системе и создает их по запросам от эмуляторов терминала. Псевдотерминал может быть запущен внутри другого псевдотерминала (так поступают, например, утилита screen или клиент ssh). Каждому псевдотерминалу в системе динамически генерируется номер и создается одноименный файл в каталоге `/dev/pts`, например, файл `/dev/pts/2` – это файл псевдотерминала под номером 2.

Из графического интерфейса эмулятор терминала можно запустить сочетанием клавиш `Alt + T` или из меню **Пуск ▸ Системные ▸ Терминал Fly**. В одном окне терминала Fly можно создавать несколько вкладок, на каждой из которых будет свой отдельный псевдотерминал. Создать вкладку можно как с помощью соответствующей кнопки в окне приложения, так и сочетанием клавиш `Ctrl + T`, а увеличить/уменьшить шрифт можно сочетанием клавиш `Ctrl + <плюс/минус>` или прокруткой колеса мыши с удержанием клавиши `Ctrl`.

Посмотреть и изменить настройки отображения терминала можно через меню **Настройка**, например, для просмотра и изменения горячих сочетаний клавиш выберите команду **Настройка ▸ Комбинации клавиш…**
Далее в курсе термин « **терминал** » будет означать окно эмулятора терминала `fly-term`.

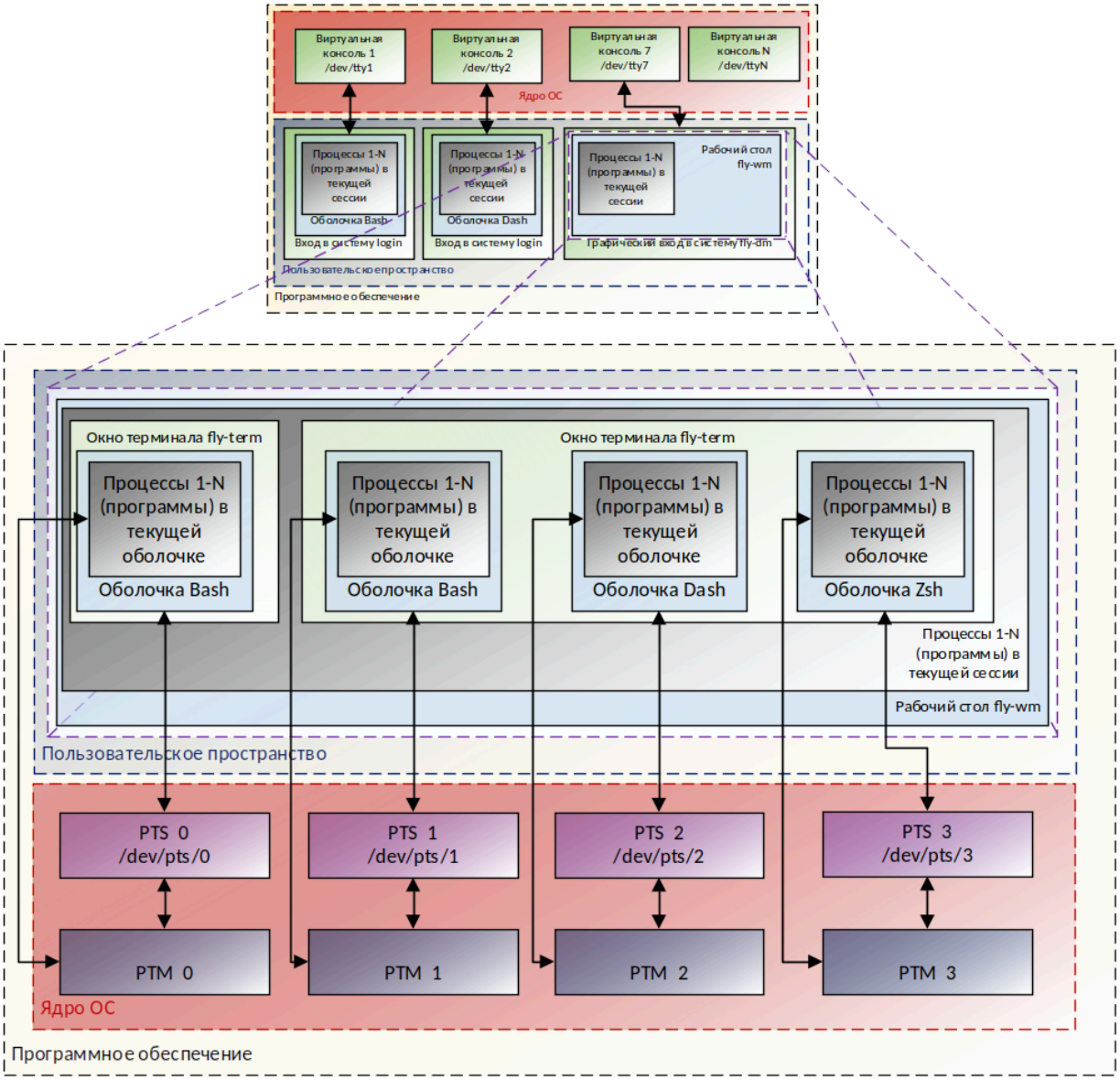
На схеме ниже отражена логика взаимодействия между программами и оболочками при работе в эмуляторе терминала (через псевдотерминал).

Немного практики:

Для того чтобы узнать номер псевдотерминала, с которым работает ваш терминал, достаточно ввести команду `tty`.

```
sa@astra:~$ tty
/dev/tty1
```

Есть несколько специальных команд, которые терминал умеет обрабатывать особым образом. Всем, кто работал в Windows, хорошо знакомо сочетание `Ctrl + C`, которое прерывает выполнение программы. В Linux оно работает точно таким же образом.



Дополненная схема с работой эмуляторов терминала и псевдотерминалов.[3]

Примечание

Команды, вызываемые с удержанием клавиши `Ctrl`, принято обозначать символом `^`, после которого следует имя клавиши, то есть горячее сочетание клавиш `Ctrl + C` — это то же самое, что `^C`.

Пример. Введем команду `du /`, которая выводит размер всех каталогов в файловой системе. Это небыстрый процесс, и, если мы не хотим ждать его завершения, то достаточно нажать `Ctrl + C`, и сигнал SIGINT завершит выполнение этой команды.

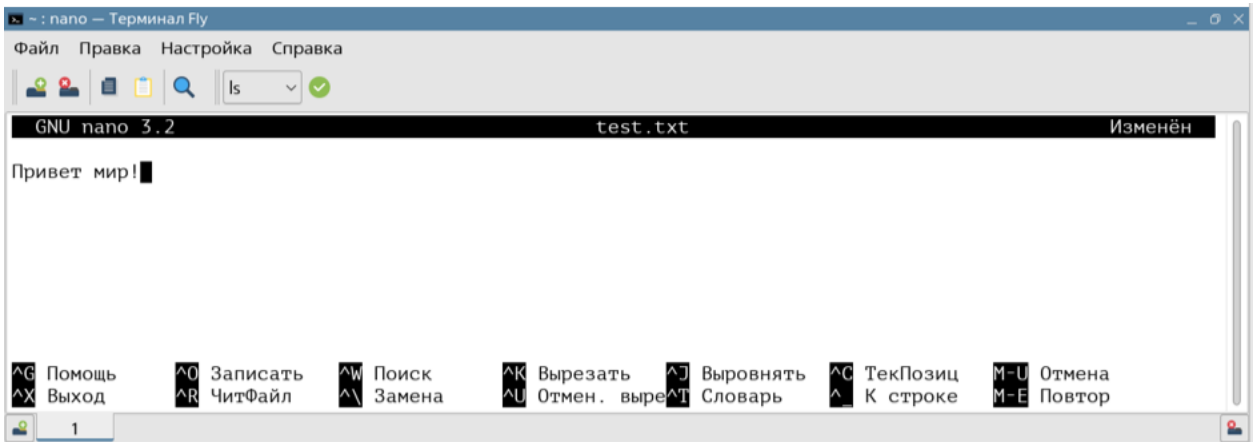
```
sa@astra:~$ du /
...
0       /proc/391/task/391/fd
0       /proc/391/task/391/fdinfo
du: невозможно прочитать каталог '/proc/391/task/391/ns': Отказано в доступе
0       /proc/391/task/391/ns
0       /proc/391/task/391/net/stat
```

```
^C
sa@astra:~$
```

Еще полезно знать, что делает сочетание клавиш `Ctrl + Z`, т.к. его часто используют по привычке для отмены последнего действия в Windows. Если вы воспользуетесь этим сочетанием клавиш во время редактирования текста в редакторе nano, то приложение исчезнет с экрана, и вы увидите сообщение *«Используйте «fg» для возврата в nano»*.

На самом деле, сочетание `Ctrl + Z` отправляет процесс в фоновый режим, и, чтобы вернуть его обратно в интерактивный режим, вам достаточно будет ввести команду `fg`. Если в фоне находится несколько приложений, то `fg` вернет к жизни последнее из них, но вы можете вызвать команду `fg` с аргументом и явно указать номер требуемого приложения.

Пример. Введем команду `nano test.txt` и напишем в редакторе, например, «Привет мир!».



Теперь нажмем сочетание клавиш `Ctrl + Z` и получим сообщение, что процесс nano был остановлен.

```
sa@astra:~$ nano test.txt
Используйте «fg» для возврата в nano
[1]+   Остановлен        nano test.txt
sa@astra:~$
```

Используя команду `fg`, переведем процесс nano, отправленный ранее в фон, обратно в интерактивный режим. Как видим, введенный нами текст никуда не делся, то есть это тот же самый процесс.

Командная оболочка Bash

Командные оболочки

При открытии терминала запускается командная оболочка, указанная в файле `/etc/passwd`. В современных дистрибутивах основной оболочкой является `/bin/bash`. Список всех установленных в системе оболочек можно посмотреть в файле `/etc/shells`.

```
sa@astra:~$ cat /etc/shells
# /etc/shells: valid login shells
/bin/sh
```

```
/bin/bash
/usr/bin/bash
/bin/rbash
/usr/bin/rbash
/bin/dash
/usr/bin/dash
```

Приведенные значения соответствуют следующим командным оболочкам:

- **/bin/sh** – Bourne Shell, старейшая оболочка UNIX, созданная Стивом Борном в 1979 году, которая в настоящее время практически не используется. Файл `/bin/sh` в зависимости от дистрибутива является символической ссылкой на /bin/bash или на `/bin/dash` (в Astra Linux `/bin/sh` - это символическая ссылка на `/bin/bash`). Если вызвать `/bin/sh`, то запустится bash в POSIX-совместимом режиме (с опцией `--posix`). Более подробно о POSIX режиме написано на сайте Bash-POSIX-Mode.html.
- **/usr/bin/bash** – Bourne Again Shell (возрожденный Shell), усовершенствованный командный интерпретатор sh от проекта GNU, который в настоящее время является одной из самых распространенных оболочек в Linux. Этот интерпретатор используется в Astra Linux по умолчанию.
- **/bin/rbash** – это bash, но с некоторыми ограничениями. Запускается по имени rbash или bash с добавлением параметра `-r`. В такой оболочке не разрешен, например, переход в другой каталог, невозможно поменять переменные окружения SHELL, PATH, ENV или BASH_ENV. Более подробно этот режим описан в man-документации на утилиту bash (секция «Ограниченный командный интерпретатор»).
- **/bin/dash** – реализация Bourne Shell, совместимая с POSIX стандартом. Занимает всего 112 Кбайт на диске, в то время как размер bash составляет 1,1 Мбайт. По сравнению с bash имеет меньше функций для интерактивной работы в командной строке, чаще используется в скриптах, так как имеет меньший размер и высокую скорость работы по сравнению с bash.

Приглашение к вводу команд

При открытии терминала мы видим командную строку с приглашением ввода.

```
san@astra:~$
```

Вот, что нам может сказать строка приглашения:

Часть	Значение
sa	текущий пользователь, от имени которого запущена командная оболочка
@	разделитель, как в электропочте
astra	имя компьютера
:	разделитель имени и рабочей директории
~	текущая директория (символ `~` обозначает домашний каталог пользователя)
$	режим работы:

Часть	Значение
	• $ - с правами простого пользователя • # - с правами суперпользователя root

Структура и синтаксис команд

В командных оболочках мы можем выполнять команды, большинство из которых вызываются как внешние утилиты, но некоторые команды все-таки встроены в саму оболочку.

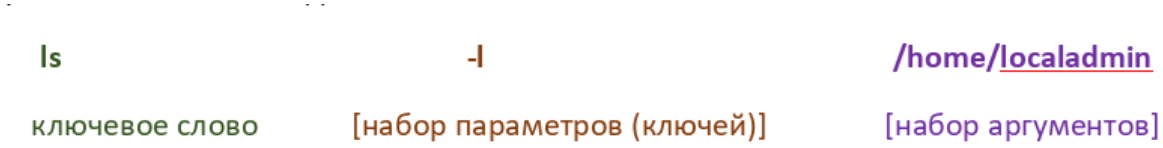
Например, команда `echo` по умолчанию является встроенной командой, поэтому при выполнении `type echo` мы увидим «echo — это встроенная команда bash».

Но если отключить эту встроенную команду `enable -n echo`, то команда `type` будет выдавать уже «echo является /usr/bin/echo», т.е. для выполнения `echo` будет вызываться уже внешняя утилита, см. пример:

```
sa@astra:~$ type echo
echo — это встроенная команда bash
sa@astra:~$ enable -n echo
saadmin@astra:~$ type echo
echo является /usr/bin/echo
sa@astra:~$
```

Список всех встроенных команд, доступных в оболочке, можно посмотреть с помощью команды `help`.

Рассмотрим синтаксис команд Bash:



- **Ключевое слово** может содержать путь до бинарного файла утилиты. Например, `ls` и `/usr/bin/ls` — это одно и то же. Если полный путь не задан, то оболочка выполнит поиск по системным каталогам, которые указаны в переменной PATH.
- **Параметры и аргументы** являются необязательными, их может вообще не быть. Для указания ключей в Linux одновременно используют три разных синтаксиса:
  - **синтаксис POSIX** – перед ключом ставится один символ дефиса, а сам ключ представляет собой строчную или прописную букву латинского алфавита. Ключи могут группироваться, когда после одного дефиса указывается сразу несколько ключей без пробелов. Например, ps -aux.
  - **синтаксис BSD** – перед ключом не ставится символ дефиса. Например, ps aux.

– **синтаксис GNU** – используются длинные ключи, перед которыми ставится двойной дефис. Например, `ps --sort cmd`. Длинные ключи могут дублировать короткие, но часто бывает, что между ними нет аналогов.

– **Но есть и исключения** из этих правил, например, команда `find` использует ключи: *-name*, *-type*, *-size* и др.

- Ключи и аргументы являются регистрозависимыми, то есть ключ «-a» и «–A» — это два совершенно разных ключа. В этом Linux сильно отличается от Windows, что многих сначала сильно путает.
- **Лишние пробелы** в командах игнорируются.

Примечание

В переменной окружения $PATH содержится список каталогов, в которых система будет искать исполняемый файл. В Windows этот механизм работает схожим образом.

```
sa@astra:~$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
```

Несколько базовых команд Bash

Давайте познакомимся с несколькими наиболее часто употребимыми командами:

- `whoami` (кто я?) — команда выводит на экран текущего пользователя, от имени которого запущен терминал:

```
sa@astra:~$ whoami
sa
```

- `echo` (эхо) — команда для отображения строки текста:

```
sa@astra:~$ echo "Привет"
Привет
```

- `pwd` (Print Working Directory, напечатать рабочую директорию) — команда для вывода на экран текущего каталога:

```
sa@astra:~$ pwd
/home/sa
```

- `cd` (Change Directory – изменить директорию) — команда для перехода по дереву каталогов. Выполним команду `cd /` и перейдем в корень файловой системы, а с помощью команды `cd -` мы вернемся обратно в каталог, в котором были:

```
sa@astra:~$ cd /
sa@astra:/$ pwd
/
sa@astra:/$ cd -
/home/sa
```

Окружение пользователя

При запуске оболочка инициализирует *окружение пользователя* — область памяти, в которой хранится набор переменных, с помощью которых осуществляется управление программами.

Для инициализации окружения оболочка bash просматривает и запускает команды из определенных файлов, находящихся в домашней директории пользователя и в системной директории /etc.

Существует несколько вариантов запуска bash:

- **Интерактивный режим оболочки входа** (interactive login shell). Если пользователь подключается к удаленной машине по ssh, заходит на машину через локальную консоль или запускает команду `su -`, то после ввода логина и пароля запускается командная оболочка bash, которая просматривает инструкции из файлов в следующем порядке: `/etc/profile`, `~/.bash_profile`, `~/.bash_login`, `~/.profile`.
-

Примечание

В Astra Linux сразу после установки отсутствуют файлы `~/.bash_profile` и `~/.bash_login`.

- **Интерактивный режим без входа в систему** (interactive non-login shell). Если пользователь уже находится в графической оболочке или в текстовой консоли, то при открытии нового окна терминала запускается оболочка bash, которая читает и выполняет инструкции из файла `~/.bashrc`.
- **Неинтерактивный режим оболочки входа** (non-interactive login shell). Если мы запускаем bash-скрипт, то оболочка смотрит на переменную окружения BASH_ENV и выполняет инструкции, которые находятся в указанном файле.
- **Неинтерактивный режим без входа в систему** (non-interactive non-login shell). Если выполнить команду `bash -c command`, то в этом режиме также выполняются инструкции из файла, заданного переменной окружения BASH_ENV.

Переменные в Bash

Переменная — это ссылка на участок памяти, в котором содержится информация для управления приложениями. Эта информация может быть считана, записана и перезаписана неограниченное количество раз.

Следует различать имя переменной (например, var1) и ссылку на её значение $var1. Если нам нужно определить переменную, то мы используем синтаксис `var1=5`, а если нам нужно вывести значение переменной на экран, то это делается так: `echo $var1`.

Типы переменных

Переменные в Bash можно разделить на три группы:

1. **переменные среды** (переменные окружения, глобальные переменные)
2. **пользовательские переменные**
3. **особые переменные** (позиционные)

Переменные среды (окружения)

Переменные среды заданы в файлах `/etc/.profile`, `~/.profile`, `~/.bashrc`, `~/.bash_profile` и инициализируются при загрузке оболочки.

Переменные окружения принято писать прописными буквами. Для просмотра всех переменных среды, доступных в текущей оболочке, можно воспользоваться командой `env` или `printenv`.

Вот некоторые из таких переменных:

- **$BASH** - полный путь до исполняемого файла командной оболочки Bash.
- **$BASH_VERSION** - версия Bash.
- **$CDPATH** - хранит пути поиска каталога (используется при вводе команды `cd имя_каталога` без слэша).
- **$CLASSPATH** - содержит список каталогов для поиска файлов классов Java и архивов Java.
- **$HOME** - домашний каталог текущего пользователя.
- **$HOSTNAME** - хранит имя компьютера.
- **$HISTSIZE** - количество событий, хранимых в истории за 1 сеанс.
- **$HISTFILE** -расположение файла истории событий.
- **$HISTFILESIZE** - количество событий, хранимых в истории между сеансами.
- **$IFS** - хранит символы, являющиеся разделителями команд и параметров (по умолчанию - пробел, табуляция и новая строка).

- **$LANG** - текущая установка локализации, которая позволяет настроить командную оболочку для использования в различных странах и на различных языках.
- **$OSTYPE** - содержится описание операционной системы.
- **$PATH** - список каталогов для поиска команд и приложений, когда полный путь к файлу не задан. Когда пользователь вводит команду, система будет проверять каталоги в указанном здесь порядке при поиске исполняемого файла.
- **$PS1** - PS1 используется как основная строка приглашения (в нашем случае *[sa@astra ~]$*).
- **$PS2** - PS2 используется как вторичная строка приглашения.
- **$PROMPT_COMMAND** - эта команда должна быть выполнена до отображения строки приглашения Bash.
- **$PWD** - полный путь к текущему рабочему каталогу.
- **$OLDPWD** - предыдущий рабочий каталог. Эта информация сохраняется оболочкой, чтобы выполнять переход к предыдущему каталогу с помощью команды `cd -`.
- **$SHELL** - полный путь к текущей командной оболочке.
- **$USER** - имя текущего пользователя.

Пользовательские переменные

При запуске оболочки никаких пользовательских переменных в ней нет. Пользовательская переменная появляется, только если пользователь сам объявит переменную, присвоив ей какое-либо значение.

Примечание

Пользовательская переменная будет доступна только в пределах процесса оболочки, в которой она была объявлена, однако её можно превратить в переменную среды командой `export`, тогда она станет доступна и для дочерних процессов, запускаемых оболочкой.

Позиционные переменные

При вызове команды или сценария с аргументами имя команды и ее аргументы являются позиционными переменными. Позиционными они называются, потому что внутри сценария обращение к ним происходит по позиции в командной строке. Позиционные переменные мы рассмотрим в лабораторной, посвященной сценариям Bash.

Действия над переменными

Рассмотрим несколько базовых действий над переменными.

Объявление переменной и присвоение ей значения

У объявления переменной или присвоения ей значения следующий синтаксис:

```
var1="Переменная1"
```

Если значение переменной не содержит пробелов, то кавычки можно опустить, в противном случае они обязательны:

```
sa@astra:~$ var1=Переменная1
sa @astra:~$ var2="Переменная     2"
sa@astra:~$ var3=Переменная 3
bash: 3: команда не найдена
```

Пробелы между именем переменной, знаком равно и её значением недопустимы:

```
sa @astra:~$ var3 ="Переменная 3"
bash: var3: команда не найдена
sa@astra:~$ var3= "Переменная 3"
bash: Переменная 3: команда не найдена
```

Значение одной переменной можно присвоить другой `var3=$var1`:

```
sa@astra:~$ var3="$var1"
```

Вывод значения переменной

При выводе переменной необходимо использовать не имя этой переменной, а ссылку на её значение `echo $var1` (символ **$** перед ней):

```
sa@astra:~$ echo "$var1"
Переменная1
sa@astra:~$ echo var1
var1
```

Если в значении переменной между словами более одного пробела, то её значение необходимо обернуть в кавычки, иначе пробелы в выводе «пропадут»:

```
sa@astra:~$ echo "$var2"
Переменная     2
sa @astra:~$ echo $var2
Переменная 2
```

Внутри одинарных кавычек, а также при экранировании символа $ не производится подстановка значений переменных:

```
sa@astra:~$ echo '$var1'
```

```
$var1
sa@astra:~$ echo \$var1
$var1
```

Обнуление и удаление переменной

Для обнуления значения переменной достаточно присвоить ей пустое значение:

```
sa@astra:~$ var1=
```

Однако это не удалит саму переменную. Для удаления переменной предназначена команда `unset <имя переменной>`:

```
sa@astra:~$ unset var1
```

Превращение пользовательской переменной в переменную среды

Для превращения пользовательской переменной, доступной только в пределах процесса текущей оболочки, в переменную среды, которая доступна и в дочерних процессах, необходимо воспользоваться командой `export`:

```
sa@astra:~$ env | grep var2
sa@astra:~$ export var2
sa@astra:~$ env | grep var2
var2=Переменная     2
```

Если мы теперь откроем дочерний процесс bash и попытаемся вывести значения переменных, то мы увидим, что переменная var2 нам доступна, а переменная var3 – нет:

```
sa@astra:~$ echo "$var2"
Переменная     2
sa@astra:~$ echo "$var3"
Переменная 3
sa@astra:~$ bash
sa@astra:~$ echo "$var2"
Переменная     2
sa@astra:~$ echo "$var3"
sa@astra:~$ exit
```

Псевдонимы в Bash

Псевдонимы в Bash - это не что иное, как «горячие клавиши», средство, позволяющее избежать набора длинных строк в командной строке. Если, к примеру, в файл `~/.bashrc` вставить строку `alias lm="ls -l | more"`, то потом вы сможете экономить свои силы и время, набирая просто команду `lm`. Установив `alias rm="rm -i"` (интерактивный режим удаления файлов), вы сможете избежать многих неприятностей, потому что сократится вероятность удаления важных файлов по неосторожности.

Создание псевдонимов

Для создания псевдонима существует команда: `alias имя_псевдонима="значение"`

```
sa@astra:~$ alias ll="ls -la"
sa@astra:~$ ll
итого 6632
drwx------ 34 sa sa    4096 мая 24 12:19  .
drwxr-xr-x  8 root       root          4096 июл 26 10:38  ..
-rw------- 1 sa sa   26916 мая 24 12:18  .bash_history
-rw-r--r-- 1 sa sa     220 мая 16 12:06  .bash_logout
-rw-r--r-- 1 sa sa   3526 мая 16 12:06  .bashrc
```

Созданный таким образом псевдоним будет работать только в текущей консоли. Чтобы сделать его постоянным, необходимо добавить эту команду в файл `~/.bash_profile` или `~/.bashrc`.

Просмотр псевдонимов

Чтобы посмотреть все псевдонимы, необходимо использовать ключ `-p`: `alias -p`

```
sa@astra:~$ alias -p
alias ll='ls -la'
alias ls='ls --color=auto'
```

Удаление псевдонима

Для удаления псевдонима существует специальная команда `unalias <имя псевдонима>`:

```
sa@astra:~$ unalias ll
sa@astra:~$ alias -p
alias ls='ls --color=auto'
```

Автодополнение и редактирование команд

- `Tab` - это одна из наиболее часто используемых горячих клавиш, которая позволяет дополнить название команды или путь к файлу. Одинарное нажатие приводит к автоматическому дополнению команды, если возможен только один вариант. Двойное нажатие отображает на экране все возможные варианты, если их несколько.
- `Ctrl + Shift + <минус>` - это сочетание клавиш отменяет последний ввод в терминал. Можно использовать для отмены автодополнения, если выбрали не тот вариант.
- `Ctrl + T` - меняет местами два соседних символа — тот, на котором курсор, и стоящий перед ним (слева). Удобно для исправления опечаток.

Примечание

В терминале fly-term на это сочетание клавиш назначено создание новой вкладки.

- `Alt + T` - меняет местами два соседних слова — то, на котором находится курсор, и стоящее перед ним (слева).

Примечание

В графической оболочке Fly на это сочетание клавиш назначено открытие нового окна терминала.

- `Alt + Backspace` - удаляет слово, стоящее перед курсором (слева), или часть слова, если курсор находится в его середине.
- `Alt + D` - удаляет слово, стоящее после курсора (справа), или часть слова, если курсор находится в его середине.

Управление курсором в терминале

Для быстрого и комфортного перемещения по тексту в командной строке (а команды и их последовательности бывают довольно длинные), стрелок навигации может быть недостаточно. Для этого есть несколько полезных комбинаций:

- `Ctrl + A` - переход в начало строки. Аналог клавиши `Home`.
- `Ctrl + E` - переход в конец строки. Аналог клавиши `End`.
- `Ctrl + B` - перемещение курсора назад (влево) на один символ, но проще стрелками.
- `Ctrl + F` - перемещение курсора вперёд (вправо) на один символ, но проще стрелками.
- `Alt + B` - перемещение курсора назад (влево) на одно слово.
- `Alt + F` - перемещение курсора вперед (вправо) на одно слово.
- `Ctrl + XX` - пожалуй, это для самых преданных фанатов Linux. Выполняет перемещение курсора между двумя точками, причем цикличное. Как правило, это начало и конец строки. Однако, если после перемещения в начало строки, был набран какой-то текст длины m, то курсор будет перемещаться между символом в строке под номером m и n-m, где m – текущая длина строки.

Работа с историей команд Bash

Все команды, которые вы набираете в окне терминала, попадают в историю команд. Если ввести команду и нажать стрелку вверх, то последняя выполненная команда появится в строке ввода, и вы сможете её выполнить повторно сразу или после того, как отредактируете. В командных оболочках Windows это работает таким же образом.

История команд хранится как в оперативной памяти, так и в файле `.bash_history`, который расположен в домашней папке пользователя. В Astra Linux по умолчанию в историю команд попадают команды со всех сессий и терминалов пользователя, это удобно.

Примечание

Количество строк, хранимых в истории, определяется переменными HISTSIZE и HISTFILESIZE. Первая определяет количество строк, хранимых в памяти, вторая — в

файле. Значения этих переменных по умолчанию задаются в профиле пользователя `~/.bashrc` и равны 1000 и 2000 соответственно.

В профиле задаются также опции оболочки, которыми можно управлять с помощью команды `shopt`. Например, чтобы включить сохранение истории со всех сессий и терминалов пользователя, в профиле прописана команда `shopt -s histappend`. Опциями оболочки можно управлять и с помощью команды `set`, список таких опций доступен в переменной SHELLOPTS.

Профили пользователей и работу с ними мы подробно разберем в отдельной лабораторной этого курса.

Просмотр команд в истории и передвижение по ней

- `Ctrl + P` - переход на одну команду вверх, альтернатива клавише стрелка вверх>.
- `Ctrl + N` - переход на одну команду вниз, альтернатива клавише стрелка вниз>.
- `history` - вывести на экран всю историю команд.
- `history 4` - вывести на экран 4 последних команд.

```
sa@astra:~$ history 4
57 alias -p
58 unalias ll
59 alias -p
60 history
```

- `history | grep pwd` - отфильтровать историю с помощью утилиты `grep`.

```
sa@astra:~$ history | grep pwd
51 pwd
52 man pwd
63 pwd
65 pwd
73 history | grep pwd
```

Поиск команд в истории

Команды в истории можно искать по совпадению с введенной строкой. Для этого есть два сочетания клавиш.

- `Ctrl + R` - включение режима поиска. После включения режима поиска наберите текст, который должен содержаться в искомой команде, и будет найдено первое совпадение от конца к началу истории. Для перехода к следующему совпадению нажимайте это же сочетание `Ctrl + R`. Чтобы выйти из режима поиска, нажмите `Ctrl + C`.

```
(reverse-i-search)`pwd': history | grep pwd
```

- `Ctrl + S` - поиск совпадения в обратную сторону. Поиск всегда начинается с конца истории, поэтому его следует начинать с `Ctrl + R`. Чтобы выйти из режима поиска, нажмите `Ctrl + C`.

```
sa@astra:~$
(i-search)`echo': echo "$var2"
```

Примечание

На это же сочетание `Ctrl + S` назначена «заморозка» терминала, когда вывод информации на него приостанавливается, но по умолчанию эта функция отключена. Однако, если терминал перестал выводить информацию, его можно «разморозить», нажав сочетание `Ctrl + Q`.

Выполнение команд из истории

С помощью символа «**!**» можно выполнить команду из истории по её номеру, например, `!39`:

```
sa@astra:~$ history 7
39  echo "$var3"
40  bash
41  history
42  history 5
43  history | grep pwd
44  history -d 44
45  history 7
sa@astra:~$ !39
echo "$var3"
Переменная 3
```

Это выражение может быть частью другой команды, например, `!41 | grep pwd`:

```
sa@astra:~$ !41 | grep pwd
history | grep pwd
   4  pwd
   8  pwd
  11  pwd
  15  pwd
  43  history | grep pwd
  50  history | grep pwd
```

Нумерацию команд можно вести с конца списка, используя относительные, отрицательные значения, например, `!-2`:

```
sa@astra:~$ history 3
53  history | grep pwd
54  pwd
55  history 3
sa@astra:~$ !-2
pwd
/home/sa
```

А специальным выражением **!!** можно быстро вызвать предыдущую команду еще раз:

```
sa@astra:~$ !!
pwd
/home/sa
```

Можно совместить поиск команды в истории и её выполнение. Выражением **!начало_команды** можно выполнить поиск первого совпадения по началу команды в истории и выполнить её:

```
sa@astra:~$ !echo
echo "$var3"
Переменная 3
```

А выражением **!?часть_команды?** можно выполнить поиск первого совпадения по любой части команды и выполнить её:

```
sa@astra:~$ !?$var2?
echo "$var2"
Переменная     2
```

Очистка истории команд

При выполнении команд, содержащих конфиденциальные данные, например, пароль учетной записи, эти данные попадут в историю команд. Чтобы такого не случилось, вы можете поставить пробел перед именем команды. Если вам потребуется исключить из занесения в историю несколько команд, то вы можете временно отключить журналирование. Если же команда уже попала в историю, то вы можете ее удалить оттуда.

Для временного отключения журналирования команд необходимо выполнить команду `set +o history`, а после `set -o history`, например:

```
$ set +o history
$ somethingScript –u username –p password –a parameter
$ set -o history
```

Если конфиденциальные данные уже попали в историю команд, то их можно удалить. Можно выборочно удалять команды или очистить всю историю сразу.

- `history -d <Номер_команды_в_истории>` - удаление команды из истории по её номеру, например, `history –d 58`:

```
sa@astra:~$ history 3
57  echo "$var3"
58  echo "$var2"
59  history 3
sa@astra:~$ history -d 58
sa@astra:~$ history 4
57  echo "$var3"
58  history 3
59  history -d 58
60  history 4
```

- `history -c` - полная очистка истории команд

```
sa@astra:~$ history -c
sa@astra:~$ history
   1  history
```

Удаление команд из истории можно произвести, отредактировав файл `.bash_history` напрямую. В текстовом редакторе можно выборочно менять его содержимое, а для полного удаления содержимого можно выполнить команду `cat /dev/null > ~/.bash_history`.

Примечание

обычный пользователь может просматривать и редактировать только свою историю команд, однако суперпользователь root может просматривать и редактировать историю команд всех пользователей в системе.

Запуск нескольких команд за раз

Оболочка Bash позволяет нам запускать несколько команд за раз, управляя при этом порядком их выполнения.

- Линейное выполнение – несколько команд будут выполняться поочередно друг за другом, независимо от результата выполнения друг друга. Для этого их необходимо разделять символом точки с запятой « **;** »:

```
sa@astra:~$ echo -n "Привет ";whoami
Привет sa
```

Ключ «n» в команде `echo` был использован, чтобы после вывода слова “Привет” не вводился невидимый символ перевода каретки и имя пользователя было выведено в той же самой строке.

- Линейное выполнение **зависимых** команд – следующая команда будет выполнена только в случае успешного завершения предыдущей. В этом случае разделителем команд будет оператор двойной амперсанд «**&&**» (логическое и):

```
sa@astra:~$ echo -n "Привет, " && whoami
Привет, sa

sa@astra:~$ WrongCommand && echo "Все работает!"
bash: WrongCommand: команда не найдена
sa@astra:~$
```

- Нелинейное выполнение зависимых команд – следующая команда будет выполнена только в случае, если предыдущая завершилась с ошибкой. В этом случае разделителем команд будет оператор «||» (логическое или):

```
sa@astra:~$ WrongCommand || echo "Что-то пошло не так!"
bash: WrongCommand: команда не найдена
Что-то пошло не так!
sa@astra:~$
```

- Параллельное выполнение независимых команд. В этом случае команды необходимо взять в скобки, а скобки разделить символом « **&** »:

```
sa@astra:~$ (echo "первое,")&(echo "второе,")&(echo "компот");echo " и можно все сразу!"
[5] 13473
[6] 13474
первое,
компот
второе,
[5]   Завершён        ( echo "первое," )
и можно все сразу!
[6]   Завершён        ( echo "второе," )
```

Стандартные потоки и перенаправление ввода/вывода

Если вы когда-нибудь работали с cmd или .bat файлами (батниками) в Windows, то, вероятно, сталкивались с конструкцией «2>». Ну, а с конструкциями «>», «>>» уж точно знакомы все. В Linux все работает таким же образом.

В Linux существуют 3 стандартных потока, которые обозначаются положительным целым числом:

- `0` – стандартный ввод (stdin) используется для передачи данных от пользователя (обычно это ввод данных с клавиатуры или из файла) к программе.
- `1` – стандартный вывод (stdout) используется для передачи данных, сгенерированных во время работы программы. По умолчанию данные выводятся на экран, но могут быть перенаправлены.
- `2` – вывод ошибок (stderr), которые могут возникнуть при выполнении программы. Ошибки выводятся отдельным потоком по умолчанию на экран, но могут быть перенаправлены.

Из стандартного потока ввода программа может только читать данные, а два других потока используются только для записи. Данные выводятся на экран, а считываются с клавиатуры, так как стандартные потоки ассоциированы с терминалом пользователя. Потоки можно подключать к файлам, программам и даже устройствам. В Bash такая операция называется перенаправлением.

Существует несколько способов перенаправления:

- `< file` – использовать файл как источник данных для стандартного потока ввода.
- `> file` – направить стандартный поток вывода в файл. Если файл не существует, он будет создан, если существует — перезаписан сверху. **Внимание! Все содержимое файла будет безвозвратно удалено!**

- `>>file` – направить стандартный поток вывода в файл. Если файл не существует, он будет создан, а если существует, то данные будут добавлены в конец файла.
- `2>file` – направить стандартный поток ошибок в файл.
- `2>>file` – направить стандартный поток ошибок в файл. Если файл не существует, он будет создан, если существует — данные будут дописаны к нему в конец.
- `&>file` или `>&file` – направить стандартный поток вывода и стандартный поток ошибок в файл.

Другая форма записи: `>file 2>&1`.

Разделение потоков позволяет, например, записывать вывод программы в один файл, а ошибки в другой.

Примеры:

- перенаправление результата выполнения команды в файл.

```
sa@astra:~$ echo "Привет!" > hello.txt
sa@astra:~$ cat hello.txt
Привет!
```

В примере использована команда `cat`, которая копирует стандартный поток ввода в стандартный поток вывода. Её часто применяют для просмотра содержимого файлов, объединения нескольких файлов в один и записи короткого текста в файл.

- перенаправление результата выполнения команды в конец файла.

```
sa@astra:~$ echo "Как дела?" >> hello.txt
sa@astra:~$ cat hello.txt
Привет!
Как дела?
```

- передача содержимого в команду в качестве аргументов.

```
sa@astra:~$ cat < hello.txt
Привет!
Как дела?
```

Конвейер

Конвейер (*англ. pipe*) – очень мощный инструмент, позволяющий передавать результаты работы одних команд в виде аргументов другим командам. Для этого используется оператор | (вертикальная черта). Тем, кто работал в PowerShell, этот инструмент должен быть хорошо знаком.

При его использовании стандартный вывод команды слева перенаправляется на стандартный ввод команды справа. При этом количество команд и перенаправлений не ограничено, что позволяет строить длинные цепочки команд, например, `cmd1 | cmd2 | .... | cmdN`. При этом конвейер обеспечивает асинхронное выполнение команд с использованием буферизации ввода/вывода. То есть нам не нужно дожидаться полного завершения первой команды, чтобы началось выполнение второй — первая команда будет передавать свои результаты второй команде по мере готовности.

Пример:

`ls` – команда выводящая содержимое текущей директории. Выполним эту команду:

```
sa@astra:~$ ls
Desktop    script                  Видео      Изображения    Шаблоны
Desktops   SystemWallpapers        Документы  Музыка
hello.txt  VBoxLinuxAdditions.run  Загрузки   Общедоступные
```

Если мы хотим вывести на экран только файлы (или каталоги), содержащие в имени «hello», мы можем использовать команду `grep`, фильтрующую строки по выбранному шаблону, передав в неё результат выполнения команды `ls` через конвейер.

Выполним команду `ls | grep "hello"`:

```
sa@astra:~$ ls | grep "hello"
hello.txt
```

Работа с буфером обмена в консоли

В терминале и, что важнее, в консоли мы можем работать с буфером обмена. Вот комбинации для работы с буфером:

- `Ctrl + W` - вырезает слово, стоящее перед курсором (слева) и копирует его в буфер обмена.
- `Ctrl + U` - вырезает текст с начала строки до положения курсора и помещает его в буфер обмена.
- `Ctrl + K` - вырезает и копирует в буфер обмена все символы в строке после курсора (справа).
- `Ctrl + Y` - вставляет текст из буфера обмена перед курсором.

Форматирование текста в консоли

В Linux, в отличие от Windows, регистр имеет большое значение – имена файлов и папок, команды, ключи являются регистрозависимыми. Для упрощения работы с регистром можно использовать комбинации:

- `Alt + U` - меняет регистр на верхний для всех символов после курсора (справа) до конца текущего слова.
- `Alt + L` - меняет регистр на нижний для всех символов после курсора (справа) до конца текущего слова.
- `Alt + C` - меняет регистр символа под курсором на верхний и переводит курсор в конец слова.

Управление отображением информации

- `Ctrl + L` - очистка терминала от текста, альтернатива команде `clear`, аналог cls в Windows.
- `Ctrl + S` - приостановка вывода данных в окно терминала программами. Программа при этом продолжает работать.

Примечание

По умолчанию эта функция отключена. Для её включения можно выполнить команду `stty ixon`, однако тогда перестанет работать обратный поиск по истории команд, назначенный на то же сочетание.

- `Ctrl + Q` - отмена приостановки вывода данных в окно терминала (отмена `Ctrl + S`).

Практические задания

Задание 1.

1. Определите файл tty текущей сессии псевдотерминала.
2. Отправьте эхо на файл псевдотерминала. Для этого откройте новый терминал, отправьте приветствие.
3. Выведите на экран переменные $COLUMNS и $LINES.
4. Запустите `sleep 9000` и через некоторое время нажмите `Ctrl + C` для прерывания команды.
5. Откройте утилиту `mc`, посмотрите иерархию файлов в mc и попробуйте закрыть ее нажатием `Ctrl + C`.
6. Почему `mc` не закрывается через `Ctrl + C`?

Задание 2.

1. Прежде чем начать работать, узнайте, в какой рабочей директории вы находитесь.
2. Измените текущую директорию на etc.

   Что изменилось в строке ввода команд?

3. Проверьте еще раз рабочий каталог.
4. Выведите список всех объектов командой `ls`.
5. Посмотрите на имя хоста, выведите командой `cat` на экран hostname.

   Ответьте на вопрос: изменился ли файл после выполнения `cat`?

   Куда команда `cat` вывела содержимое файла?

   Безопасна ли команда `cat`?

6. Сделайте копию hostname перед изменением перенаправления STDOUT.

   ```
   cat hostname > ~/hostname.old
   ```

   Куда сохранился файл hostname.old?

7. Проверьте, как сохранился бекап:

   ```
   cat < ~/hostname.old
   ```

8. Поменяйте имя хоста с помощью редактора nano:

   ```
   nano /etc/hostname
   ```

   Ответьте на вопрос: почему подчеркивает красным цветом [File „/etc/hostname“ is unwritable]?

   Закройте редактор с помощью `Ctrl + X`. И если при выходе из nano редактор запросит сохранить изменения, то нужно нажать `N` и `Enter`.

9. Повторите команду с повышенными правами.

```
$ sudo !!
```

Комбинация символов **!!** подставляет предыдущую команду

Важно

Выполнять предыдущую команду `sudo !!` можно, только если вы знаете, какая команда была до этого. Иначе можно сломать что-нибудь важное в системе. **Используйте аккуратно, зная, что root может всё.**

1. Измените текст на dc-1 (будущее название хоста).
2. Мы написали что-то неправильно и на автомате нажали `Ctrl + Z`.Привычка Windows отменять введенный текст этим сочетанием.
   Нажмите `Ctrl + Z`.

   Этим действием мы отправили процесс на паузу управляющей последовательностью `Ctrl + Z`.
3. Верните процесс из фонового режима командой `fg`.
4. Сохраните нужный нам текст dc-1, нажимая `Ctrl + O` и Enter.
5. Закройте редактор nano с помощью `Ctrl + X`.

6. Чтобы сменилось название хоста, выполните перезагрузку.
7. После смены имени хоста будут возникать проблемы с отображением `sudo`, и для этого надо изменить хост в `/etc/hosts`.

Задание 3.

1. Выведите приветствие текущему пользователю. Где можно использовать приветствие и переменную $USER?
2. Посмотрите, какие есть общие глобальные переменные окружения.
3. Выведите все переменные текущей сессии.
4. Найдите с помощью `grep` фильтра в текстовых данных конвейером только LINES или COLUMNS.
5. Попробуйте перезапустить пк в текущем пользователе без root.

Ответьте, почему не получилось, посмотрев на переменную окружения $PATH:

1. Посмотрите, какие бинарные файлы может запускать обычный пользователь.
2. Посмотрите, какие файлы может запускать root пользователь. Для этого выполните вход в сессию root пользователем.

Ответьте на вопрос: Почему пользователь не может найти команду `reboot`, хотя она есть у root пользователя?

Вопросы

1. Как называется устройство, которое может отправлять команды ЭВМ и выводит на экран полученный результат?

2. Какая папка отвечает за конфигурационные файлы?

3. Какая управляющая последовательность завершает операцию?

4. Какая команда выводит список файлов и каталогов текущей директории?

5. Какими текстовыми редакторами можно редактировать файл?

6. Какой командой можно получить справку на любую команду?

7. Какой командой можно перенаправить стандартный вывод в файл hosts.bak?

8. В какой переменной хранится список каталогов для запуска исполняемых файлов?

9. Какой поток данных передается по конвейеру?

10. Какая команда отображает историю команд?

11. Какой файл содержит профиль текущего пользователя?