Лабораторная работа №7
Работа с профилями пользователей. Переменные окружения

Введение

Из этой лабораторной работы вы узнаете как именно Linux создает окружение пользователя и почему набор инициализационных файлов далеко не ограничивается `.bashrc`. Также поговорим о порядке инициализации рабочего стола fly-wm и его конфигурационных файлах, чтобы вы могли применить эти знания и для настройки графического окружения.

Профили пользователя в Linux

Под профилем пользователя в широком смысле понимают весь его домашний каталог целиком, а в более узком — набор скриптов, которые выполняются каждый раз при запуске оболочки и позволяют определить индивидуальные настройки окружения пользователя.

При создании локального пользователя создается его домашний каталог, в который копируются файлы из папки `/etc/skel`, задающие основу профиля (*сокращение от англ. skeleton — каркас*). Большинству администраторов Linux хорошо известен такой файл как `~/.bashrc`. Однако он является далеко не единственным скриптом, определяющим окружение пользователя. Также окружение пользователя определяют `~/.bash_aliases`, `~/.bash_login`, `~/.bash_logout` ,`~/.bash_profile`, `~/.profile` и многие другие.

Порядок инициализации профиля

Во-первых, скрипты инициализации бывают разные:

- Часть скриптов инициализации создали для того, чтобы можно было определить «параметры компьютера» в целом, т.е. их действие распространяется на всех пользователей системы сразу. Такие скрипты называют **общесистемными** (System-wide Startup Files), и обычно они находятся в каталоге `/etc`.
- Другая часть предназначена для кастомизации окружения конкретного пользователя. Они называются **пользовательскими** файлами (User-specific startup files) и находятся как раз в домашних каталогах.
- И, конечно же, не забываем про сценарии выхода **из системы** (logout).

Во-вторых, для возможности тонкой настройки системы порядок инициализации профиля зависит от контекста использования оболочки. Выделяют следующие режимы ее работы:

По способу аутентификации пользователя:

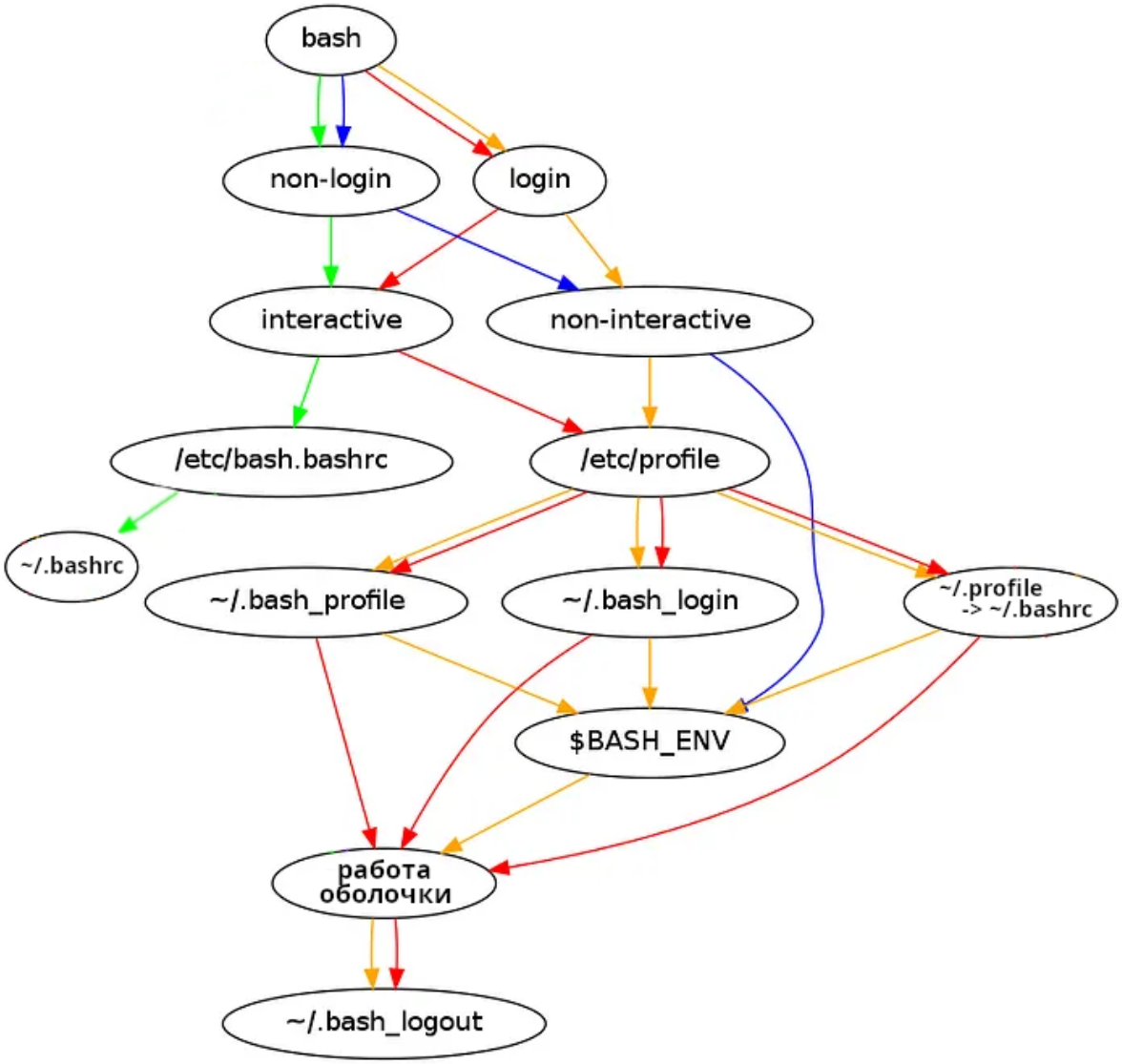
- **Вход в систему с вводом учетных данных** (login) — режим, для входа в который пользователь должен пройти аутентификацию, т.е. система запросит у него учетные данные. В этом режиме окружение пользователя начинает формироваться с «чистого листа». Пример: вход с помощью утилит `login`, `su`, подключение по `ssh`.
- **Вход в сеанс без ввода учетных данных** (non-login) — режим, для входа в который от пользователя не требуется прохождение аутентификации. В этом режиме на вход скриптов инициализации подается набор переменных, которые были определены в рамках входа в систему с вводом учетных данных. Примеры: запуск `fly-term` из графики, создание новой вкладки в этом приложении, вход в новый экземпляр оболочки из текущей сессии командой bash без параметров.

По способу взаимодействия пользователя с оболочкой:

- **Интерактивный** или командный (interactive) — режим, в котором пользователь может вводить команды вручную и наблюдать в окне результаты их выполнения.
- **Неинтерактивный** или пакетный (non-interactive) — режим, в котором оболочка не взаимодействует с пользователем и завершает свою работу сразу же после выполнения указанных команд. Например, локальное и удаленное выполнение строки команд или целого скрипта:
  - `bash -c 'echo $PWD'`,
  - `bash myscript.sh`,
  - `./bash myscript.sh`,
  - `ssh user@host 'echo $PWD'`.

Ну, и в-третьих, для хранения определенных настроек выделяют отдельные файлы. Например, команды, создающие псевдонимы, выносят в отдельный файл `bash_aliases`.

Теперь можно перейти к рассмотрению алгоритма (см. рисунок).



Алгоритм инициализации профиля пользователя в оболочке Bash

Интерактивный вход без аутентификации

Когда вы открываете новое окно терминала `fly-term` или вкладку в окне этого приложения, порядок инициализации скриптов должен быть таким, как показан на схеме зелеными стрелочками.

Скрипт /etc/bash.bashrc является общесистемным скриптом. В нем могут устанавливаться размер окна, внешний вид приглашения командной строки, настройки автодополнения, обработка выполнения несуществующих команд.

Следует понимать, что скрипт `/etc/bash.bashrc` начинает формировать окружение не «с чистого листа», а получает на вход ряд переменных, которые были установлены ранее в рамках «интерактивного входа в систему с аутентификацией».

Скрипт ~/.bashrc является пользовательским. В нем устанавливаются настройки сохранения истории команд HISTCONTROL, HISTSIZE, HISTFILESIZE и др, подключается файл `~/.bash_aliases` со списком псевдонимов команд.

Параметры, устанавливаемые с помощью этого скрипта, предназначены только для интерактивного режима работы оболочки. При установке пакета bash-completion из этого файла дополнительно подключаются скрипты этого пакета. Кстати, буквы rc в конце имени файла означают «run command».

Скрипт ~/.bash_aliases является пользовательским и определяет перечень псевдонимов команд. Алиасы могут быть определены в скрипте `~/.bashrc`, но правильнее это делать в специальном файле :file:` ~/.bash_aliases`, который подключается из `~/.bashrc`. Приведем пару псевдонимов команд в качестве примера:

```
alias ll='ls -l'
alias la='ls -la'
```

Примечание

Графическая оболочка Fly при запуске приложений совершает дополнительные обращения к системе, которые приводят к вызову скрипта `~/.bashrc` в неинтерактивном режиме, поэтому в начале скрипта `~/.bashrc` на эту тему есть дополнительная проверка.

Неинтерактивный вход без аутентификации

Если вы запускаете скрипты по имени, передаете файл скрипта на вход приложению bash или запускаете отдельные команды через ключ `-c`, то порядок инициализации будет таким, как показано на схеме синими стрелочками, т.е. будет запускаться скрипт, определенный в переменной BASH_ENV.

Так же, как и при интерактивном входе без аутентификации, скрипт из BASH_ENV начинает формировать окружение не «с чистого листа», а получает ряд переменных, которые были установлены ранее в рамках «интерактивного входа в систему с аутентификацией».

Интерактивный вход с аутентификацией

Если вы выполняете вход в систему из консоли `Ctrl + Alt + F1`, подключаетесь через SSH или выполняете интерактивный вход утилитами `login`, `su`, `sudo -i` и др., то порядок инициализации будет таким, как показано на схеме красными стрелочками:

Скрипт /etc/profile является общесистемным скриптом. В нем устанавливаются инструкции для настройки переменных окружения PATH, PS1, и вызывается скрипт `/etc/bash.bashrc`. Дополнительно вызываются скрипты из каталога `/etc/profile.d/*`, чтобы пользовательские приложения могли расширить набор инициализируемых переменных.

Скрипт ~/.profile является пользовательским скриптом и позволяет переопределить те переменные окружения, которые были заданы на глобальном уровне. Например, разработчики программного обеспечения могут расширить значение переменной PATH и включить в нее каталоги своих проектов.

Обратите внимание, что на втором шаге загрузки Bash выбирает один из трех файлов: `~/.bash_profile`, `~/.bash_login` или `~/.profile`. Изначально в оболочке Bourne был только `~/.profile`, но разработчики Bash добавили обработку собственных файлов, чтобы можно было определять дополнительные инструкции, специфичные для этой оболочки. В настоящий момент эта возможность используется не во всех дистрибутивах, и в Astra Linux, к примеру, этих файлов нет, поэтому запускается скрипт `~/.profile`.

Неинтерактивный вход с аутентификацией

Остался последний случай, когда мы выполняем команды в пакетном режиме, но с прохождением аутентификации. Порядок инициализации будет таким, как показано на схеме желтыми стрелочками, т.е. дополнительно будет запускаться скрипт, определенный переменной BASH_ENV.

Сценарии выхода из системы

Из скрипта `~/.bash_logout` можно, например, удалить какие-нибудь временные файлы приложения. Но не забывайте, что он срабатывает только при полном выходе из системы. Выход из сессии, например, закрытие вкладки окна терминала, не приводит к запуску этого скрипта.

Примечание

В профиле пользователя есть файл `~/.bash_history`, который представляет собой обычный текстовый файл со списком ранее выполненных команд. Этот файл загружается в момент входа в систему, и в него выгружается история команд при заполнении буфера и корректном завершении сессии.

Профиль рабочего стола fly

Выбор режима рабочего стола Fly выполняется в меню «Тип сессии» в окне графического входа в систему (утилита `fly-dm`). По умолчанию предусмотрено несколько режимов, но администратор системы может добавить новые режимы, например, для систем с низкими характеристиками производительности или удаленных терминалов можно создавать режим fly-light и т.д.

Для создания нового режима необходимо добавить файл (файлы) сессии с расширением `*.desktop` в директории `/usr/share/fly-dm/sessions` и создать соответствующие конфигурационные файлы для `fly-wm`.

При входе через fly-dm выставляется переменная `DESKTOP_SESSION=имя_режима`, например: *fly*, *fly-desktop*, *fly-tablet*. Данная переменная является именем ярлыка сессии из каталога */usr/share/fly-dm/sessions* (но без расширения `.desktop`), которая указывает на тип сессии. Например: **DESKTOP_SESSION=fly** — десктопный; **DESKTOP_SESSION=fly-tablet** — планшетный.

Данное имя сессии добавляется как суффикс «.$DESKTOP_SESSION» к базовому имени конфигурационного файла и используется для выбора конфигурационных файлов менеджера окон fly-wm в соответствии с типом сессии. Если тип сессии десктопный, т.е. DESKTOP_SESSION=fly, то конфигурационные файлы остаются без суффикса для обратной совместимости.

Существуют следующие конфигурационные файлы в `/usr/share/fly-wm/` и соответствующие им файлы в `~./fly/`.

```
sa@astra:~$ cat ~/.fly/en.fly-wmrc
;update
[Variables]
;Dont edit this main file, edit files included below

ImagePath       =/usr/share/fly-wm/images:/usr/share/fly-wm/images:/usr/share/icons
SoundPath       =/usr/share/fly-wm/sounds:/usr/share/fly-wm/sounds
XmmPath         =/usr/share/fly-wm/keymaps:/usr/share/fly-wm/keymaps

include ~/.fly/paletterc
include ~/.fly/sessrc
include /usr/share/fly-wm/keyshortcutrc
include ~/.fly/keyshortcutrc
include ~/.fly/apprc
include ~/.fly/theme/current.themerc
include ~/.fly/en.miscrc
```

Файл `en.fly-wmrc` – главный файл, из которого вызываются остальные:

- `~/.fly/paletterc` – цветовая схема

- `~/.fly/sessr` – задание переменных, определяющих некоторые параметры сессий (использование менеджера сессий, диалоги при выключении компьютера, значения по умолчанию, новый вход и так далее)
- `/usr/share/fly-wm/keyshortcutrc` – глобальные настройки комбинаций клавиш (для всех пользователей)
- `~/.fly/keyshortcutrc` – индивидуальные настройки комбинаций клавиш
- `~/.fly/apprc` – настройка атрибутов для приложений
- `~/.fly/theme/current.themerc` – настройки текущей темы рабочего стола
- `~/.fly/en.miscrc` – конфигурация различных меню: Пуск, контекстные меню рабочего стола и его элементов (панель задач, область уведомлений и так далее)

Такие же файлы существуют и для других режимов работы, это:

- `~/.fly/en.fly-wmrc.fly-kiosk`
- `~/.fly/en.fly-mini`
- `~/.fly/en.fly-wmrc.fly-mobile`
- `~/.fly/en.fly-tablet`
- `~/.fly/en.fly-wmrc.fly-tablet-kiosk`

Вызов функций рабочего стола

Вызовом команды `fly-wmfunc <имя_функции> [<параметры>]` можно управлять элементами рабочего стола, например, чтобы свернуть все окна, необходимо выполнить команду `fly-wmfunc FLYWM_MINIMIZE_ALL`, а чтобы восстановить их – `fly-wmfunc FLYWM_RESTORE_ALL`.

```
sa@astra:~$ fly-wmfunc FLYWM_MINIMIZE_ALL
sa@astra:~$ fly-wmfunc FLYWM_RESTORE_ALL
```

По ссылке доступен список и описание функций fly-wm: Описание функций Fly-wm.

Принудительное обновление пользовательских конфигурационных файлов при входе пользователя

Для того чтобы при каждом входе пользователя пользовательская конфигурация автоматически перезаписывалась конфигурацией, заданной администратором системы, нужно указать параметр «;update» в первой строке файлов в каталоге /usr/share/fly-wm/:

```
;update
```

Файлы, в которых указан этот параметр, будут заменять соответствующие пользовательские файлы при входе пользователя в графическую сессию.

Подробнее о принудительном обновлении конфигурационных файлов при входе пользователя и их отдельных параметров вы можете почитать по ссылке: Управление пользовательскими конфигурациями оконного менеджера fly-wm.

Переменные окружения

Впервые переменные окружения (среды) появились еще в Unix и с тех пор были включены во все современные операционные системы, включая Linux, MacOS, Windows. В отличие от Windows, в операционной системе Linux не только системные, но и многие пользовательские приложения используют переменные окружения в своей работе.

Уровни переменных окружения

Переменные могут быть определены на нескольких уровнях:

- **На уровне сеанса**

Переменная определяется с помощью команды `export` и действует в рамках текущего сеанса. Например, вы можете «на лету» переопределить переменную PATH с помощью команды `export PATH="$PATH:~/.local/bin"`, но это значение не станет доступно новым экземплярам оболочки.

- **На уровне пользователя**

В тех случаях, когда необходимо установить переменные для конкретного пользователя, их определяют в скриптах из его домашнего каталога, например, `~/.bashrc`. При запуске оболочки этим пользователем переменные будут устанавливаться автоматически в соответствии с порядком инициализации профиля.

- **На уровне системы**

Если же необходимо, чтобы переменные окружения устанавливались для всех пользователей компьютера, их нужно определить в файле `/etc/profile`. При запуске оболочки эти переменные будут устанавливаться автоматически в соответствии с порядком инициализации профиля.

Команды для работы с переменными окружения

Команда printenv

Команда `printenv` показывает все переменные окружения командной оболочки: и те, что были установлены в скриптах инициализации, и те, которые пользователь установил сам с помощью команды `export`:

```
sa@astra:~$ printenv
SHELL=/bin/bash
SESSION_MANAGER=local/astra:@/tmp/.ICE-unix/1031,unix/astra:/tmp/.ICE-unix/1031
WINDOWID=0
QT_ACCESSIBILITY=1
PAM_TTY=/dev/tty7
PAM_MAC_SAVED_MACLABEL=0:63:0x0:0x0!
FLY_CLIENT_LOG=.xsession-errors
ICEAUTHORITY=/run/user/1000/.ICEauthority
LANGUAGE=
SSH_AUTH_SOCK=/tmp/ssh-AYZ5N561WsN8/agent.1031
SHELL_SESSION_ID=376914a280ce4bd5be5992191d649d80
XDM_MANAGED=method=classic
DESKTOP_SESSION=fly
SSH_AGENT_PID=1172
GTK_MODULES=gail:atk-bridge
XDG_SEAT=seat0
PWD=/home/sa
XDG_SESSION_DESKTOP=fly
```

Команда set

Команда `set` выводит более подробный список переменных окружения, локальных переменных оболочки, а также различных функций, используемых командным интерпретатором. Вывод команды большой, поэтому выведем только первые 20 строк:

```
sa@astra:~$ set | head -n 20
BASH_ALIASES=()
BASH_ARGC=([0]="0")
BASH_ARGV=()
BASH_CMDS=()
BASH_COMPLETION_VERSINFO=([0]="2" [1]="8")
BASH_ENV=/home/sa/.bashrc
BASH_LINENO=()
BASH_SOURCE=()
BASH_VERSINFO=([0]="5" [1]="0" [2]="3" [3]="1" [4]="release" [5]="x86_64-pc-linux-gnu"
)
BASH_VERSION='5.0.3(1)-release'
COLORFGBG='15;0'
COLUMNS=86
DBUS_SESSION_BUS_ADDRESS=unix:abstract=/tmp/dbus-HJnCKQoJo5,guid=125da4b11b4797b35b8c6
3c267d16107
DESKTOP_SESSION=fly
DIRSTACK=()
DISPLAY=:0
DM_CONTROL=/var/run/xdmctl
EUID=1000
```

Команда export

Команду `export` мы уже использовали. Она устанавливает значение переменной окружения в рамках текущей сессии.

Команда env

Без параметров команда выводит информацию, аналогичную printenv, но основной смысл ее не в этом. Команда позволяет запускать приложения с измененным окружением без переопределения переменных в рамках текущей сессии.

Еще можно переопределить язык запускаемой команды, например, прочитать английскую версию bash в manpages (пустая переменная означает локаль по умолчанию *en_US.UTF-8*):

```
sa@astra:~$ env LANG= man bash
```

Но также можно задать локаль принудительно зная ее iso-код в файле `/etc/locale.gen`:

```
sa@astra:~$ locale
LANG=ru_RU.UTF-8
LANGUAGE=
LC_CTYPE="ru_RU.UTF-8"
LC_NUMERIC="ru_RU.UTF-8"
LC_TIME="ru_RU.UTF-8"
...
sa@astra:~$ env LANG=en_US.UTF-8 man bash
sa@astra:~$ env LANG=ru_RU.UTF-8 man bash
```

Примеры использования переменных окружения

Переменная PATH

Если необходимо добавить новую директорию, где буду храниться пользовательские скрипты и программы, потребуется изменить переменную PATH. Создадим директорию `scripts` в домашней папке пользователя и файл `myscript.sh` внутри нее. Для простоты `myscript.sh` выводит сообщение «Hello!». Для текущего сеанса достаточно выполнить команду `export`. После изменения PATH скрипт будет запускаться без указания полного пути к скрипту:

```
sa@astra:~$ mkdir ~/scripts
sa@astra:~$ cat <<EOF> ~/scripts/myscript.sh
> #!/usr/bin/env bash
> echo "Hello!"
> EOF
sa@astra:~$
sa@astra:~$ chmod +x ~/scripts/myscript.sh
sa@astra:~$ echo "$PATH"
/home/sa/.local/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
sa@astra:~$ ./myscript.sh
```

```
bash: ./myscript.sh: Нет такого файла или каталога
sa@astra:~$ export PATH="$PATH:~/scripts"
sa@astra:~$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games: ~/scripts
sa@astra:~$ myscript.sh
Hello!
```

После завершения текущей командной оболочки переменная PATH не сохранится. Чтобы сделать изменения постоянными, необходимо добавить новое значение PATH для пользователя в файлы инициализации `~/.bash_profile`, `~/.profile` или `~/.bashrc`. Добавим в файл `~/.bashrc`, так как при запуске оболочки из графического режима инструкции из `~/.bash_profile`, `~/.profile` не будут просмотрены командной оболочкой.

Откроем новую вкладку в терминале и выполним команды:

```
sa@astra:~$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
sa@astra:~$ echo 'export PATH="$PATH:~/scripts"' >> ~/.bashrc
sa@astra:~$ bash
sa@astra:~$ echo "$PATH"
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games: ~/scripts
sa@astra:~$ myscript.sh
Hello!
```

Если требуется изменить PATH для всех пользователей системы, тогда необходимо добавить изменения в глобальный файл инициализации `/etc/profile`. Чтобы скрипт `myscript.sh` стал общим, переименуем его в *script.sh* и положим в директорию */opt/scripts*. Для демонстрации примера создадим отдельную директорию и отредактируем `/etc/profile`. Так как директория `/usr/local/bin` включена в PATH всех пользователей, то достаточно было бы положить свои программы и скрипты в эту директорию без изменения `/etc/profile`. Изменения `/etc/profile` требуют права root. После внесения изменений перезайдем на машину под пользователем sa.

```
sa@astra:~$ sudo -i
[sudo] пароль для sa:
root@astra:~# mkdir /opt/scripts
root@astra:~# cp /home/sa/scripts/myscript.sh /opt/scripts/script.sh
root@astra:~# echo 'export PATH="$PATH:/opt/scripts"' >> /etc/profile
root@astra:~# su - sa
sa@astra:~$ script.sh
Hello!
sa@astra:~$ echo "$PATH"
/home/sa/.local/bin:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/opt/scripts:/home/sa/.local/bin:~/scripts
sa@astra:~$ whereis script.sh
script: /usr/bin/script /opt/scripts/script.sh /usr/share/man/man1/script.1.gz
```

Как мы видим *file*:*/opt/scripts* добавилось для всех пользователей в переменную PATH

Переменные истории команд

Переменные истории команд HISTCONTROL, HISTFILE, HISTSIZE, HISTFILESIZE, HISTIGNORE, HISTTIMEFORMAT.

HISTCONTROL – определяет, как команды сохраняются в истории команд. Принимает следующие значения (несколько значений могут быть указаны через двоеточие):

- **ignorespace** – если команда начинается с пробела, она не сохраняется в истории.
- **ignoredups** – если введенная команда содержится в истории, она не сохраняется.
- **ignoreboth** – включает в себя обе опции **ignorespace** и **ignoredups**.
- **erasedups** – если введенная команда содержится в истории, вся ее предыдущая история удаляется и команда сохраняется заново.

В Astra Linux значение по умолчанию `HISTCONTROL=ignoreboth`. Если значение не задано или указано неправильное значение, все введенные команды сохраняются как есть. Чтобы команды, начинающиеся с пробела, сохранялись в истории, необходимо поменять значение HISTCONTROL. После изменения команды с пробелом будут сохраняться. В примере команда `ping 88.77.8.8` будет сохранена.

```
sa@astra:~$ HISTCONTROL=ignoredups
sa@astra:~$ ping 88.77.8.8
PING 88.77.8.8 (88.77.8.8) 56(84) bytes of data.
64 bytes from 88.77.8.8: icmp_seq=1 ttl=52 time=87.2 ms
64 bytes from 88.77.8.8: icmp_seq=2 ttl=52 time=136 ms
64 bytes from 88.77.8.8: icmp_seq=3 ttl=52 time=144 ms
64 bytes from 88.77.8.8: icmp_seq=4 ttl=52 time=126 ms
64 bytes from 88.77.8.8: icmp_seq=5 ttl=52 time=202 ms
64 bytes from 88.77.8.8: icmp_seq=6 ttl=52 time=88.3 ms
64 bytes from 88.77.8.8: icmp  seq=7 ttl=52 time=93.7 ms
64 bytes from 88.77.8.8: icmp_seq=8 ttl=52 time=88.3 ms
^C
--- 88.77.8.8 ping statistics ---
8 packets transmitted, 8 received, 0% packet loss, time 12ms
rtt min/avg/max/mdev = 87.208/120.628/202.400/37.749 ms

sa@astra:~$ history -a
sa@astra:~$ tail -2 .bash_history
ping 88.77.8.8
history -a
```

HISTFILE – определяет файл, в который записывается история. По умолчанию это `~/.bash_history`. Если задать другой файл, то надо иметь в виду, что только у пользователя должно быть право на чтение и запись этого файла.

```
sa@astra:~$ ls -l .bash_history
-rw------- 1 sa sa 43857 сен 15 21:37 .bash_history
```

HISTSIZE – определяет максимальное число строк, которое будет запомнено командной оболочкой. Если HISTSIZE равен 0, история команд не будет сохраняться в памяти оболочки. Если HISTSIZE меньше нуля, будут запоминаться все команды. Если число строк будет превышено, из файла будут удалены старые строки и добавлены новые так, что размер файла не превысит HISTSIZE.

HISTFILESIZE – определяет максимальный размер файла истории в строках. Если размер равен 0, из файла удаляются все строки. Если размер задан в виде нечисловых символов или равен числу меньше 0, размер файла не ограничен.

Чтобы сделать хранение истории неограниченным, можно присвоить пустые значения:

- HISTSIZE=
- HISTFILESIZE=

HISTIGNORE – определяет шаблоны команд, которые не попадут в память оболочки для сохранения в файле истории. Несколько шаблонов могут быть перечислены через двоеточие.

```
sa@astra:~/scripts$ HISTIGNORE="ls -l"
sa@astra:~/scripts$ ls -l
итого 252
. . .
sa@astra:~/scripts$ history -a
sa@astra:~/scripts$ tail -2 ~/.bash_history
HISTIGNORE="ls -l"
history -a
```

Но при этом команда `ls -la` попадет в историю.

```
sa@astra:~/scripts$ ls -la
итого 440
. . .
sa@astra:~/scripts$ history -a
sa@astra:~/scripts$ tail -2 ~/.bash_history
ls -la
history -a
```

Чтобы исключить все команды, начинающиеся с ls -l, можно добавить символ «звездочка» (*).

```
sa@astra:~/scripts$ HISTIGNORE="ls -l*"
sa@astra:~/scripts$ ls -la
итого 440
. . .
sa@astra:~/scripts$ history -a
sa@astra:~/scripts$ tail -2 ~/.bash_history
HISTIGNORE="ls -l*"
history -a
```

HISTIGNORE удобен для того, чтобы исключить из истории стандартные команды (ls, exit и другие) и команды, содержащие чувствительную информацию.

- **HISTTIMEFORMAT** – формат сохранения истории. По умолчанию в истории команд не показывается дата, так как не задана переменная HISTTIMEFORMAT. Для отображения даты необходимо определить её формат, например, HISTTIMEFORMAT=%H:%M:%S%d-%m-%Y

```
sa@astra:~$ cd scripts
sa@astra:~/scripts$ history | tail -n 2
710  cd scripts
 711  history | tail -n 2
sa@astra:~/scripts$ echo "$HISTTIMEFORMAT"

sa@astra:~/scripts$
```

Как мы видим формат имеет пустое значение, а теперь установим переменную для формата времени истории:

```
sa@astra:~/scripts$ HISTTIMEFORMAT='%H:%M:%S %d-%m-%Y '
```

Проверим вывод команды истории `history`:

```
sa@astra:~/scripts$ history -a
sa@astra:~/scripts$ history | tail -n 3
713  11:57:32 14-03-2025 HISTTIMEFORMAT='%H:%M:%S %d-%m-%Y '
 714  11:57:39 14-03-2025 history -a
 715  11:57:48 14-03-2025 history | tail -n 3
```

Переменная PROMPT_COMMAND

По умолчанию история команд сохраняется при нормальном завершении сеанса, например, командой `exit` или сочетанием клавиши `Ctrl + D`. Однако, при аварийном завершении команды не сохранятся. С помощью PROMPT_COMMAND история может быть сохранена немедленно.

```
sa@astra:~$ PROMPT_COMMAND="history -a;$PROMPT_COMMAND"
sa@astra:~$ echo "Hello"
Hello
sa@astra:~$ tail -3 .bash_history
PROMPT_COMMAND="history -a;$PROMPT_COMMAND"
#1741942818
echo "Hello"
sa@astra:~$ echo "Who am I"
Who am I
sa@astra:~$ tail -3 .bash_history
tail -2 .bash_history
#1741942895
echo "Who am I"
sa@astra:~$
```

Еще с помощью этой переменной можно, например, добавить дату в приглашении командной строки.

```
sa@astra:~$ PROMPT_COMMAND="echo -n [$(date +%H:%M:%Y)]"
[12:02:2025]sa@astra:~$
```

Переменные PS0, PS1, PS2, PS3, PS4

Переменные PS0, PS1, PS2, PS3, PS4 для настройки вида приглашения командной строки.

PS0 – задает значение, которое выводится перед выводом команды. По умолчанию не определено.

```
sa@astra:~$ PS0="Команда запущена пользователем \u на машине \h:\n\n"
sa@astra:~$ echo Hello
```

```
Команда запущена пользователем user на машине astra:

Hello
```

PS1 – определяет вид приглашения командной строки. По умолчанию для пользователей Astra Linux командная строка выделена разными цветами в виде строки `sa@astra:~$`.

```
sa@astra:~$ echo "$PS1"
\[\e]0;\u@\h: \w\a\]${debian_chroot:+($debian_chroot)}\u@\h:\w\$
```

- **`\[\e]0;\u@\h: \w\a\]` - задает название окна терминала (отображается в верхней части окна) в виде `sa@astra:~`, где:**

- **\u** – имя пользователя,
- **\h** – имя машины,
- **\w** – текущий каталог. Символы **[** и **]** обозначают начало и конец последовательности непечатных символов.
- **\e]0** – это управляющая ESCAPE последовательность, которая определяет действие для операционной системы, в данном случае поменять заголовок окна.
- **\a** – обозначает ASCII символ звукового сигнала (bell).
- `${debian_chroot:+($debian_chroot)}` – Если задана переменная debian_chroot, добавляет ее к приглашению.
- **$** - выводит символ $.

Командная строка может быть выделена разными цветами, например, значение переменной PS1:

```
PS1="\[\e]0;\u@\h:\w\a\]${debian_chroot:+($debian\_chroot)}\[\033[01;91m\][\D{%H:%M:%S %d.%m.%Y}]
\[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]\$ "
```

- `\[\033[01;32m\]\u@\h` – выводит в определенном цвете логин пользователя и имя машины, разделенные символом @. Управляющая Esc последовательность [\033[01;32m\] указывает операционной системе вывести следующий за ней текст в зеленом цвете.
- `[\033[00m\]:` – выводит символ : в белом цвете.
- `[\033[01;34m\]\w` – выводит текущую директорию в синем цвете.
- `\[\033[00m\]\$` – выводит символ $ в белом цвете.

Можно добавить отображение даты к текущему виду приглашения в виде `\[\033[01;91m\]\D{%H:%M:%S %d.%m.%Y}` , где `\D` – дата в формате, заданном в фигурной скобке. Дата будет окрашена в красный цвет.

```
sa@astra:~$ PS1="\[\e]0;\u@\h:\w\a\]${debian_chroot:+($debian\_chroot)}\[\033[01;91m\]
[\D{%H:%M:%S %d.%m.%Y}] \[\033[01;32m\]\u@\h\[\033[00m\]:\[\033[01;34m\]\w\[\033[00m\]
\$ "
[12:20:23 14.03.2025] sa@astra:~$
```

Примечание

Оттенков цветов очень много. Для примера здесь приведена таблица с цветами, а здесь – информация о специальных символах.

PS2 – устанавливает символ продолжения строки. Если строка не помещается на экране целиком, по умолчанию это символ `>`.

```
sa@astra:~$ echo "$PS2"
>
sa@astra:~$ PS2=">>>"
sa@astra:~$ ping 88.77.8.8 \
>>>-c 10 -l 1 \
>>>-W 10
```

PS3 – выводит строку приглашения при использовании в скриптах команды `select`. Если запустить скрипт сначала без установки PS3, приглашение выбора выглядит в виде #?.

```
sa@astra:~$ echo "$PS3"
```

Создадим такой скрипт `user_ps3.sh`:

```
#!/usr/bin/env bash

select i in пон вт ср exit
do
  case $i in
    пон) echo "Понедельник";;
    вт) echo "Вторник";;
    ср) echo "Среда";;
    exit) exit;;
  esac
done
```

Запустим его через bash:

```
sa@astra:~$ bash user_ps3.sh
1) пон
2) вт
3) ср
4) exit
#? 1
Понедельник
```

Если определить PS3, вид приглашения изменится.

```
#!/usr/bin/env bash

PS3="Выберите день недели: "

select i in пон вт ср exit
do
  case $i in
    пон) echo "Понедельник";;
    вт) echo "Вторник";;
    ср) echo "Среда";;
    exit) exit;;
  esac
done
```

Запустим его повторно:

```
sa@astra:~$ bash user_ps3.sh
1) пон
2) вт
3) ср
4) exit
Выберите день недели: 1
Понедельник
```

PS4 – определяет вид приглашения при включении режима отладки в скрипте. Переименуем и изменим немного предыдущий скрипт `user_ps3.sh`. Добавим ключ `-x`, который включит режим отладки (вывод дополнительной служебной информации) для поиска ошибок в скрипте. Также установим переменную PS4 равным `PS4="$0.$LINENO "`, где `$0` – имя скрипта, а `$LINENO` – номер строки.

```
sa@astra:~$ echo "$PS4"
+
```

Создадим еще один скрипт для проверки PS4 `user_ps4.sh`

```
#!/usr/bin/env bash

PS3="Выберите день недели: "
PS4="$0.$LINENO "
select i in пон вт ср exit
 do
  case $i in
    пон) echo "Понедельник";;
    вт) echo "Вторник";;
    ср) echo "Среда";;
    exit) exit;;
  esac
done
```

Проверим его работу через bash:

```
sa@astra:~$ bash -x user_ps4.sh
+ case $- in
+ return
+ PS3='Выберите день недели: '
+ PS4='user_ps4.sh.4 '
user_ps4.sh.4 select i in пон вт ср exit
1) пон
2) вт
3) ср
4) exit
Выберите день недели: 1
user_ps4.sh.4 case $i in
user_ps4.sh.4 echo Понедельник
Понедельник
Выберите день недели: 2
user_ps4.sh.4 case $i in
user_ps4.sh.4 echo Вторник
Вторник
```

К выводу скрипта добавилась дополнительная информация.

Мы рассмотрели некоторые примеры использования переменных окружения в командной оболочке bash, которая используется по умолчанию в Astra Linux. Документация по всем переменным доступна в `man bash` или `info bash`.

Практические задания

Задание 1.

1. Создайте директорию `tmp_files` в `/etc/skel`.
2. Убедитесь, что в домашнем каталоге нового созданного пользователя появилась папка `tmp_files`. Войдите в систему под новым пользователем, создайте файл в директории `tmp_files`.
3. Добавьте команду очистки файлов и папок в директории `tmp_files` в `.bash_logout`. Сделайте вход в систему командой `login` и после выхода проверьте, что файлы в директории `tmp_files` исчезли.

Задание 2.

1. Создайте директорию `my_programm` в домашней папке пользователя.
2. Добавьте путь к директории `my_programm` в переменную окружения PATH. Убедитесь, что после выхода и входа в систему переменная PATH содержит путь к `my_programm`.

Задание 3.

1. Установите переменную окружения HISTTIMEFORMAT в виде «%H:%M:%S %d-%m-%Y» глобально.
2. Войдите под пользователем, введите несколько команд, убедитесь, что в выводе команды history появилась дата ввода команд.
3. Создайте файл `.bash_aliases`, добавьте псевдоним команды удаления файлов `rm`, но только с подтверждением перед удалением. Проверьте, что файлы не удаляются сразу, а сначала запрашивается разрешение пользователя.

Вопросы

1. Как называется файл в Astra Linux, в котором определяются настройки окружения для конкретного пользователя?

2. В каком файле определяется переменная окружения PATH для всех пользователей?

3. В какой директории хранятся шаблоны файлов и директорий для создаваемых пользователей?

4. Если пользователь создает свою переменную окружения, в каком файле он может ее определить?

5. Что делает команда printenv?

6. Какой файл определит настройки окружения пользователя, если в домашней директории пользователя лежат сразу четыре файла - .bash_profile, .bash_login, .profile, .bashrc?

7. Какой командой можно создать переменную окружения в текущем сеансе пользователя?

8. С помощью какой команды можно запустить программу, передав ей измененные значения переменных окружения?

9. В каком отдельном файле можно определить псевдонимы команд для конкретного пользователя?

10. Какая командная оболочка используется в Astra Linux по умолчанию?