

## Часть 2. Bourne Shell aka POSIX sh.

Влад 'mend0za' Шахов  
Linux & Embedded Team Leader

Linux & Embedded Department



# Что такое Unix shell?

## Что такое Unix shell? (Назойливый повтор)

- Обычная программа, запускающаяся после входа в систему
- Интерактивный командный интерпретатор
- Язык программирования
- Платформа интеграции (для утилит)
- Сотни разных реализаций (bash, ksh, zsh, tcsh, ...)
- Масса различных диалектов



# Shell. Ключевые понятия - 1

## Определения

- Приглашение командной строки (CMD PROMPT):



# Shell. Ключевые понятия - 1

## Определения

- Приглашение командной строки (CMD PROMPT):

\$, #, user@host:~\$



# Shell. Ключевые понятия - 1

## Определения

- Приглашение командной строки (CMD PROMPT):

`$, #, user@host:~$`

- Команда:

`whoami; top; exit`



# Shell. Ключевые понятия - 1

## Определения

- Приглашение командной строки (CMD PROMPT):

`$, #, user@host:~$`

- Команда:

`whoami; top; exit`

- Параметр:

`man bash; who am i`



# Shell. Ключевые понятия - 1

## Определения

- Приглашение командной строки (CMD PROMPT):

`$, #, user@host:~$`

- Команда:

`whoami; top; exit`

- Параметр:

`man bash; who am i`

- Ключ (1 символ):

`ls -a; ls -al; ls -a -l /tmp/`



# Shell. Ключевые понятия - 1

## Определения

- Приглашение командной строки (CMD PROMPT):

`$, #, user@host:~$`

- Команда:

`whoami; top; exit`

- Параметр:

`man bash; who am i`

- Ключ (1 символ):

`ls -a; ls -al; ls -a -l /tmp/`

- Длинный ключ (GNU-style):

`ls --version`





## Shell. Ключевые понятия - 2

### Картинка для закрепления

```
.fetchmailrc.sample
mend0za@ak112:/home/mend0za/tmp: ls -l etc/
mend0za@ak112:/home/mend0za/tmp: ls -la etc/
mend0za@ak112:/home/mend0za/tmp: ls -la etc/
..
nmap-2.0.11
mutt-ak112-1000-1479-223877113186584578
1034x1200-dsc06692.jpg mutt.html
1600x1404-dsc06703.jpg openvpn.tgz
build ppp.tgz
disqus.js sankercup2012-final-protocol-signed.pdf
etc template.sh
.fetchmailrc.sample
mend0za@ak112:/home/mend0za/tmp: ls -la etc/
total 16
drwxr-xr-x 4 mend0za mend0za 4096 Окт 19 09:25 .
drwxr-xr-x 5 mend0za mend0za 4096 Янв 8 16:07 ..
drwxr-xr-x 2 mend0za mend0za 4096 Июн 6 2012 openvpn
drwxr-xr-x 3 mend0za mend0za 4096 Окт 19 09:25 ppp
mend0za@ak112:/home/mend0za/tmp: su
Password:
su: Authentication failure
mend0za@ak112:/home/mend0za/tmp: su
Password:
root@ak112:/home/mend0za/tmp: export PS1="# "
# pwd
/home/mend0za/tmp
# whoami
root
#
```

команды / ключи  
/ параметры .

приглашение  
командной  
строки  
( \$ # > )

работа с  
переменными



## Приёмы эффективной работы

Как в Shell работать быстро?



## Приёмы эффективной работы

### Как в Shell работать быстро?

- 1 автодополнение путей и команд
- 2 история команд
- 3 редактирование командной строки



## Приёмы эффективной работы

### Автодополнение путей и команд - 1

# Волшебная кнопка - TAB



---

<sup>1</sup>Только у BASH и ZSH (если настроены)

# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды



---

<sup>1</sup>Только у BASH и ZSH (если настроены)

## Приёмы эффективной работы

### Автодополнение путей и команд - 1

## Волшебная кнопка - TAB

- Имя команды

Пример: `mys [TAB] _co [TAB]`



---

<sup>1</sup>Только у BASH и ZSH (если настроены)

# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды

Пример: `mys[TAB]_co[TAB]`

Результат: `mysql_convert_table_format`

8 vs 26



---

<sup>1</sup>Только у BASH и ZSH (если настроены)

## Приёмы эффективной работы

### Автодополнение путей и команд - 1

## Волшебная кнопка - TAB

- Имя команды

Пример: `mys[TAB]_co[TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов



---

<sup>1</sup>Только у BASH и ZSH (если настроены)



## Приёмы эффективной работы

### Автодополнение путей и команд - 1

## Волшебная кнопка - TAB

- Имя команды

Пример: `mys[TAB]_co[TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов

Пример: `ls /u[TAB]lo[TAB]sh[TAB]/ca[TAB]`



---

<sup>1</sup>Только у BASH и ZSH (если настроены)

# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды

Пример: `mys[TAB]_co[TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов

Пример: `ls /u[TAB]lo[TAB]sh[TAB]/ca[TAB]`

Результат: `ls /usr/local/share/ca-certificates/`

16 vs 36



---

<sup>1</sup>Только у BASH и ZSH (если настроены)

# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды

Пример: `mys [TAB]_co [TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов

Пример: `ls /u [TAB]lo [TAB]sh [TAB]/ca [TAB]`

Результат: `ls /usr/local/share/ca-certificates/`

16 vs 36

- Параметры и ключи <sup>1</sup>

---

<sup>1</sup>Только у BASH и ZSH (если настроены)



# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды

Пример: `mys [TAB]_co [TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов

Пример: `ls /u [TAB]lo [TAB]sh [TAB]/ca [TAB]`

Результат: `ls /usr/local/share/ca-certificates/`

16 vs 36

- Параметры и ключи <sup>1</sup>

Пример: `apti [TAB]--a [TAB]sh [TAB]core [TAB] [ENTER]`

---

<sup>1</sup>Только у BASH и ZSH (если настроены)



# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды

Пример: `mys [TAB]_co [TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов

Пример: `ls /u [TAB]lo [TAB]sh [TAB]/ca [TAB]`

Результат: `ls /usr/local/share/ca-certificates/`

16 vs 36

- Параметры и ключи <sup>1</sup>

Пример: `apti [TAB]--a [TAB]sh [TAB]core [TAB] [ENTER]`

Результат: `aptitude --assume-yes show coreutils`

16 vs 37

---

<sup>1</sup>Только у BASH и ZSH (если настроены)



# Приёмы эффективной работы

## Автодополнение путей и команд - 1

### Волшебная кнопка - TAB

- Имя команды

Пример: `mys [TAB]_co [TAB]`

Результат: `mysql_convert_table_format`

8 vs 26

- Пути и имена файлов

Пример: `ls /u [TAB]lo [TAB]sh [TAB]/ca [TAB]`

Результат: `ls /usr/local/share/ca-certificates/`

16 vs 36

- Параметры и ключи <sup>1</sup>

Пример: `apti [TAB]--a [TAB]sh [TAB]core [TAB] [ENTER]`

Результат: `aptitude --assume-yes show coreutils`

16 vs 37

---

<sup>1</sup>Только у BASH и ZSH (если настроены)



## Приёмы эффективной работы

### Автодополнение путей и команд - 2

Единственный вариант подстановки:

TAB дополняет сразу



## Приёмы эффективной работы

### Автодополнение путей и команд - 2

Единственный вариант подстановки:

TAB дополняет сразу

Несколько вариантов подстановки?

Ещё больше волшебства - 2 кнопки TAB!

2xTAB - список вариантов подстановки





## Приёмы эффективной работы

### Автодополнение путей и команд - 3

# Примеры

- `apt[TAB][TAB]`
- `aptitude -- [TAB] [TAB]`<sup>2</sup>
- `ls /[TAB][TAB]`<sup>3</sup>

---

<sup>2</sup>Только для BASH и ZSH

<sup>3</sup>Можно использовать вместо команды ```ls```



# Приёмы эффективной работы

## История команд

### Просмотр истории

- ``Up'' и ``Down'' - вперёд-назад



# Приёмы эффективной работы

## История команд

### Просмотр истории

- ``Up'' и ``Down'' - вперёд-назад
- ``Ctrl+R'' - интерактивный поиск в истории



# Приёмы эффективной работы

## История команд

### Просмотр истории

- ``Up'' и ``Down'' - вперёд-назад
- ``Ctrl+R'' - интерактивный поиск в истории
- повторно ``Ctrl+R`` - искать дальше



## Приёмы эффективной работы

### Редактирование командной строки

#### Emacs editing mode <sup>4</sup>

- ``Left'' и ``Right'' - вперёд-назад по текущей строке



---

<sup>4</sup>Только KSH-совместимые: bash, zsh, pdksh, mksh, etc

## Приёмы эффективной работы

### Редактирование командной строки

#### Emacs editing mode <sup>4</sup>

- ``Left'' и ``Right'' - вперёд-назад по текущей строке
- ``Ctrl+a'' и ``Ctrl+e'' - перейти в начало и конец строки

---

<sup>4</sup>Только KSH-совместимые: bash, zsh, pdksh, mksh, etc



## Приёмы эффективной работы

### Редактирование командной строки

#### Emacs editing mode <sup>4</sup>

- ``Left'' и ``Right'' - вперёд-назад по текущей строке
- ``Ctrl+a'' и ``Ctrl+e'' - перейти в начало и конец строки
- ``Ctrl+u'' - удалить от курсора до начала строки



---

<sup>4</sup>Только KSH-совместимые: bash, zsh, pdksh, mksh, etc

## Приёмы эффективной работы

### Редактирование командной строки

#### Emacs editing mode <sup>4</sup>

- ``Left'' и ``Right'' - вперёд-назад по текущей строке
- ``Ctrl+a'' и ``Ctrl+e'' - перейти в начало и конец строки
- ``Ctrl+u'' - удалить от курсора до начала строки
- ``Ctrl+w'' - удалить слово (от курсора до разделителя, влево)

---

<sup>4</sup>Только KSH-совместимые: bash, zsh, pdksh, mksh, etc





## Условное выполнение команд

**Код возврата (RETURN CODE):**  
результат выполнения у любой команды Shell

Shell return code:

- 0 - выполнен успешно
- не 0 - ошибка



## Условное выполнение команд

**Код возврата (RETURN CODE):**  
результат выполнения у любой команды Shell

Shell return code:

- 0 - выполнен успешно
- не 0 - ошибка

Операции над кодом возврата:

- ``&&'' - логическое И
- ``||'' - логическое ИЛИ



## Условное выполнение команд

**Код возврата (RETURN CODE):**  
результат выполнения у любой команды Shell

Shell return code:

- 0 - выполнен успешно
- не 0 - ошибка

Операции над кодом возврата:

- `&&` - логическое И
- `||` - логическое ИЛИ

Примеры:

- `cat /proc/1/environ || echo fail`
- `find /usr/share/doc -name '*.txt' && echo ok`



## Скрипты

### Shell Script, определение

Последовательность команд Shell.

Разделитель: перевод строки, ``;''



## Скрипты

### Shell Script, определение

Последовательность команд Shell.

Разделитель: перевод строки, ``;''

### shebang

`#!/something` или чем мы запускаем скрипт.

По умолчанию : `#!/bin/sh`

Всегда первая строка скрипта.

Фактически: `/bin/sh scriptname`



## Скрипты

### Shell Script, определение

Последовательность команд Shell.

Разделитель: перевод строки, ``;''

### shebang

`#!/something` или чем мы запускаем скрипт.

По умолчанию : `#!/bin/sh`

Всегда первая строка скрипта.

Фактически: `/bin/sh scriptname`

### Парадоксальные примеры

`#!/bin/rm`

`#!/bin/awk -f`

`#!/bin/less`



## Запуск скриптов

- ❶ `sh scriptname`
- ❷ `chmod +x script`  
`./script`
- ❸ из каталогов в переменной `PATH`  
`echo $PATH`  
`~/bin` (если есть)  
`/usr/local/bin`
- ❹ в текущей копии shell<sup>5</sup>  
`. ./script`  
`source script`<sup>6</sup>

---

<sup>5</sup>Остальные способы - запускают новый shell

<sup>6</sup>Несовместимо с POSIX. Происходит из ksh. Добавляет текущий каталог к списку путей



## Потоки ввода-вывода

Особенности архитектуры<sup>7</sup>:

**У каждой запущенной программы 3 потока I/O:**

- ① ввода
- ① вывода
- ② ошибок

Связаны с экраном и клавиатурой терминала.

---

<sup>7</sup>См документацию языка программирования Си





## Потоки ввода-вывода

Особенности архитектуры<sup>7</sup>:

**У каждой запущенной программы 3 потока I/O:**

- 0 ввода
- 1 вывода
- 2 ошибок

Связаны с экраном и клавиатурой терминала.

**Связаны с терминалом только по умолчанию**

shell позволяет переопределить весь ввод и вывод программы

---

<sup>7</sup>См документацию языка программирования Си



## Базовый синтаксис перенаправления

- Ввод ``<''  
sort <.bash\_history



---

<sup>8</sup>Файл затрёт новым содержанием, если он существовал ранее

## Базовый синтаксис перенаправления

- **Ввод** ``<"  
sort <.bash\_history
- **Вывод** ``>"<sup>8</sup>  
find /usr/share/doc -name ``\*.txt'' >txt-docs
- **Вывод** ``1>"  
find /usr/share/doc -name ``\*.txt'' 1>txt-docs

---

<sup>8</sup>Файл затрёт новым содержанием, если он существовал ранее



## Базовый синтаксис перенаправления

- **Ввод** ``<"  
sort <.bash\_history
- **Вывод** ``>"<sup>8</sup>  
find /usr/share/doc -name ``\*.txt'' >txt-docs
- **Вывод** ``1>"  
find /usr/share/doc -name ``\*.txt'' 1>txt-docs
- **Ошибки** ``2>"  
find /tmp 2>find.errors

---

<sup>8</sup>Файл затрёт новым содержанием, если он существовал ранее



## Базовый синтаксис перенаправления

- **Ввод** ``<"  
sort <.bash\_history
- **Вывод** ``>"<sup>8</sup>  
find /usr/share/doc -name ``\*.txt'' >txt-docs
- **Вывод** ``1>"  
find /usr/share/doc -name ``\*.txt'' 1>txt-docs
- **Ошибки** ``2>"  
find /tmp 2>find.errors
- **Вывод (дописать в конец)** ``1>>"  
find /usr/share/doc -name ``\*.txt'' >>txt-docs
- **Ошибки (дописать в конец)** ``2>>"  
find /tmp 2>>find.errors

---

<sup>8</sup>Файл затрёт новым содержанием, если он существовал ранее



## Базовый синтаксис перенаправления

- **Ввод** ``<"  
sort <.bash\_history
- **Вывод** ``>"<sup>8</sup>  
find /usr/share/doc -name ``\*.txt'' >txt-docs
- **Вывод** ``1>"  
find /usr/share/doc -name ``\*.txt'' 1>txt-docs
- **Ошибки** ``2>"  
find /tmp 2>find.errors
- **Вывод (дописать в конец)** ``1>>"  
find /usr/share/doc -name ``\*.txt'' >>txt-docs
- **Ошибки (дописать в конец)** ``2>>"  
find /tmp 2>>find.errors

---

<sup>8</sup>Файл затрёт новым содержанием, если он существовал ранее



## Расширенный синтаксис перенаправления

- **Pipe**<sup>9</sup> `cmd1 | cmd2`

Вывод `cmd1` направляется на ввод `cmd2`.

```
man bash|grep ksh
```

---

<sup>9</sup>Классика Unix



## Расширенный синтаксис перенаправления

- **Pipe**<sup>9</sup> ``cmd1 | cmd2 ''

Вывод cmd1 направляется на ввод cmd2.

```
man bash|grep ksh
```

- **Склеить потоки** ``N>&M''

В примере: посмотреть одновременно и вывод и ошибки

```
find /tmp 2>&1 | less
```

---

<sup>9</sup>Классика Unix





## Расширенный синтаксис перенаправления

- **Pipe**<sup>9</sup> `cmd1 | cmd2`

Вывод `cmd1` направляется на ввод `cmd2`.

```
man bash|grep ksh
```

- **Склеить потоки** `N>&M`

В примере: посмотреть одновременно и вывод и ошибки

```
find /tmp 2>&1 | less
```

- **"Ввод здесь"**<sup>10</sup> `<<END_MARKER`

```
sort <<EOF
```

```
oieu
```

```
ak
```

```
zf
```

```
EOF
```

---

<sup>9</sup>Классика Unix



## Практика: перенаправление, скрипты - 1

### Задание 1

- 1 Сохранить 5 последних команд из истории в файл.
- 2 Убрать номера (редактировать любым способом)
- 3 Выполнить скрипт (не делая файл исполняемым)



## Практика: перенаправление, скрипты - 1

### Задание 1

- 1 Сохранить 5 последних команд из истории в файл.
- 2 Убрать номера (редактировать любым способом)
- 3 Выполнить скрипт (не делая файл исполняемым)

### Задание 2

- 1 Добавить заголовок, говорящий о том, что файл является shell-скриптом
- 2 Сделать файл исполняемым
- 3 выполнить скрипт как исполняемый файл



## Практика: перенаправление, скрипты - 2

### Задание 3

Сделать вывод сохранённой в скрипте истории команд на экран.

Самим скриптом<sup>11</sup>.

---

<sup>11</sup> конструкция ``ввод здесь''



## Практика: перенаправление, скрипты - 2

### Задание 3

Сделать вывод сохранённой в скрипте истории команд на экран.

Самим скриптом<sup>11</sup>.

### Задание 4

Отсортировать вывод из **Задания 3** в обратном порядке.  
Не использовать временные файлы.

---

<sup>11</sup> конструкция ``ввод здесь''



## Практика: перенаправление, скрипты - 2

### Задание 3

Сделать вывод сохранённой в скрипте истории команд на экран.

Самим скриптом<sup>11</sup>.

### Задание 4

Отсортировать вывод из **Задания 3** в обратном порядке.  
Не использовать временные файлы.

### Задание 5

Отсортировать и вывести на экран содержимое скрипта (в 1 команду), используя перенаправление ввода-вывода.

---

<sup>11</sup> конструкция ``ввод здесь''



## Переменные

### Переменные:

настройки окружения пользователя для процесса <sup>12</sup>



---

<sup>12</sup>Смотри `environ(7)` о подробностях реализации

## Переменные

### Переменные:

настройки окружения пользователя для процесса <sup>12</sup>

### Какие бывают?

- встроенные (в Shell):
  - HOME - домашний каталог
  - PWD - текущий каталог
  - PATH - список каталогов, где ищут исполняемые файлы
  - PS1 - приглашение пользователя
- пользовательские



---

<sup>12</sup>Смотри `environ(7)` о подробностях реализации



## Просмотр и изменение значений переменных

- **set** - просмотр списка



## Просмотр и изменение значений переменных

- **set** - просмотр списка
- **\$HOME** - взять значение переменной



## Просмотр и изменение значений переменных

- **set** - просмотр списка
- **\$HOME** - взять значение переменной

```
echo $USER $HOME  
echo $PATH  
$SHELL
```



## Просмотр и изменение значений переменных

- **set** - просмотр списка
- **\$HOME** - взять значение переменной

```
echo $USER $HOME  
echo $PATH  
$SHELL
```

- **unset** - сброс (обнуление) значения



## Просмотр и изменение значений переменных

- **set** - просмотр списка
- **\$HOME** - взять значение переменной

```
echo $USER $HOME  
echo $PATH  
$SHELL
```

- **unset** - сброс (обнуление) значения
- **VAR1=`значение`** - установить новое значение

*Примечание:* в значении можно использовать переменные (если значение присваивается через ``")



## Волшебные виды кавычек

- **Одиночные** : 'всё как было'

```
echo 'Oops: $HOME $SHELL '
```



---

<sup>13</sup> пробел - разделитель параметров

## Волшебные виды кавычек

- **Одиночные** : 'всё как было'

```
echo 'Oops: $HOME $SHELL'
```

- **Двойные** : "раскрывает значения переменных"

```
echo "Ok: _$HOME_$SHELL"
```



## Волшебные виды кавычек

- **Одиночные** : 'всё как было'

```
echo 'Oops: $HOME $SHELL'
```

- **Двойные** : "раскрывает значения переменных"

```
echo "Ok: _$HOME_$SHELL"
```

- **Обратные** : `выполняем команду`

```
LISTING=`ls -a`  
echo "'in $PWD: $LISTING"
```

Одиночные и двойные экранируют пробелы<sup>13</sup>.

<sup>13</sup>пробел - разделитель параметров





## Примеры из жизни

```
$CHROOT_TOOL /bin/sh /tmp/`basename $hook`
```



## Примеры из жизни

```
$CHROOT_TOOL /bin/sh /tmp/`basename $hook`
```

```
ROOT_UUID=`tune2fs -l $ROOT_DEV | awk '/  
    UUID/ {print $3}'`
```



## Примеры из жизни

```
$CHROOT_TOOL /bin/sh /tmp/`basename $hook`
```

```
ROOT_UUID=`tune2fs -l $ROOT_DEV | awk '/  
UUID/ {print $3}'`
```

```
(echo unit B; echo print; echo quit) | \  
parted $FLASH_IMG | \  
awk '/^ [1-2]/ { printf "—offset=%s—  
sizelimit=%s/dev/loop%s\n", $2, $4,  
$1 }' | \  
sed 's/B//g' | \  
xargs -n 1 -l{} sh -c "losetup -v {}  
$FLASH_IMG"
```



## Практика: переменные и кавычки

**Задание 1.** Запустить скрипт с историей<sup>14</sup> в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.



---

<sup>14</sup> из предыдущего задания по перенаправлению ввода-вывода

## Практика: переменные и кавычки

**Задание 1.** Запустить скрипт с историей<sup>14</sup> в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.

**Задание 2.** Скрипт-генератор случайных чисел (RANDOM)



---

<sup>14</sup> из предыдущего задания по перенаправлению ввода-вывода

## Практика: переменные и кавычки

**Задание 1.** Запустить скрипт с историей<sup>14</sup> в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.

**Задание 2.** Скрипт-генератор случайных чисел (RANDOM)

**Задание 3.** Работа с путями для поиска команд.

- 1 создать папку `bin` в домашнем каталоге
- 2 дописать в `PATH` полный путь к созданной папке (используя переменную `HOME`)
- 3 скопировать скрипт с историей в папку `bin`
- 4 запустить скрипт, без указания пути к нему

---

<sup>14</sup> из предыдущего задания по перенаправлению ввода-вывода



## Подстановочные символы путей (Wildcards)

**Wildcards** - спецсимволы в параметрах команд, раскрываемые в путь и имя файла самим интерпретатором перед тем, как запустить команду на выполнение.



---

<sup>15</sup> об интервалах - в разделе о регулярных выражениях

## Подстановочные символы путей (Wildcards)

**Wildcards** - спецсимволы в параметрах команд, раскрываемые в путь и имя файла самим интерпретатором перед тем, как запустить команду на выполнение.

- **\*** - любое количество любых символов

```
echo /u*  
ls /u*
```





## Подстановочные символы путей (Wildcards)

**Wildcards** - спецсимволы в параметрах команд, раскрываемые в путь и имя файла самим интерпретатором перед тем, как запустить команду на выполнение.

- **\*** - любое количество любых символов

```
echo /u*  
ls /u*
```

- **[]** - символ из перечисления<sup>15</sup>

```
echo .[bp]*
```



<sup>15</sup> об интервалах - в разделе о регулярных выражениях

## Подстановочные символы путей (Wildcards)

**Wildcards** - спецсимволы в параметрах команд, раскрываемые в путь и имя файла самим интерпретатором перед тем, как запустить команду на выполнение.

- **\*** - любое количество любых символов

```
echo /u*  
ls /u*
```

- **[]** - символ из перечисления<sup>15</sup>

```
echo .[bp]*
```

- **?** - любой одиночный символ

```
echo ?i*
```



<sup>15</sup> об интервалах - в разделе о регулярных выражениях

## Практика: Wildcards

**Упражнение 1.** Вывести на экран имя файла или каталога, содержащего ровно 4 символа и начинающегося с точки



---

<sup>16</sup> вывод файлов и каталогов, включая скрытые

## Практика: Wildcards

**Упражнение 1.** Вывести на экран имя файла или каталога, содержащего ровно 4 символа и начинающегося с точки

**Упражнение 2.** -- || -- из папки /usr, содержащее 'i' или 'h'



## Практика: Wildcards

**Упражнение 1.** Вывести на экран имя файла или каталога, содержащего ровно 4 символа и начинающегося с точки

**Упражнение 2.** -- || -- из папки /usr, содержащее 'i' или 'h'

**Упражнение 3.** Написать скрипт lsa, делающий то же, что и команда `ls -a`<sup>16</sup>, не используя `ls`.



---

<sup>16</sup> вывод файлов и каталогов, включая скрытые

## Условия и ветвление. if и test(1)

if то же самое, что и test(1)



## Условия и ветвление. if и test(1)

if то же самое, что и test(1)

```
if [ -f .bash_history ]  
then  
    cat .bash_history  
fi
```

полностью идентично по результату

```
test -f .bash_history && cat .bash_history
```

if сложнее - все пробелы значащие (часть синтаксиса).



## Часто используемые унарные параметры test(1)

- **-f filepath** - проверить на существование файл

```
test -f "$FILEPATH" && rm -v $FILEPATH
```





## Часто используемые унарные параметры test(1)

- **-f filepath** - проверить на существование файл

```
test -f "$FILEPATH" && rm -v $FILEPATH
```

- **-d directorypath** - проверить на существование каталог

```
if [ -d "$HOME/bin" ]  
then  
    echo "file $HOME/bin present"  
else  
    mkdir "$HOME/bin"  
fi
```



## Часто используемые унарные параметры test(1)

- **-f filepath** - проверить на существование файл

```
test -f "$FILEPATH" && rm -v $FILEPATH
```

- **-d directorypath** - проверить на существование каталог

```
if [ -d "$HOME/bin" ]  
then  
  echo "file $HOME/bin present"  
else  
  mkdir "$HOME/bin"  
fi
```

- **-z ``value''** - строка пустая

```
test -z "$VARIABLE" && VARIABLE="value"
```



## Часто используемые унарные параметры test(1)

- **-f filepath** - проверить на существование файл

```
test -f "$FILEPATH" && rm -v $FILEPATH
```

- **-d directorypath** - проверить на существование каталог

```
if [ -d "$HOME/bin" ]  
then  
    echo "file $HOME/bin present"  
else  
    mkdir "$HOME/bin"  
fi
```

- **-z ``value''** - строка пустая

```
test -z "$VARIABLE" && VARIABLE="value"
```

- **-n ``value''** - строка ненулевая

```
test -n "$VARIABLE" && echo "$VARIABLE"
```



## Часто используемые бинарные параметры test(1)

- **STRING1 = STRING2** строки равны
- **STRING1 != STRING2** строки не равны
- **INTEGER1 -eq INTEGER2**<sup>17</sup> числа равны
- **INTEGER1 -ne INTEGER2** числа не равны
- **INTEGER1 -gt INTEGER2** число 1 больше числа 2



---

<sup>17</sup>Привет учившим ассемблер

## Практика: условия

**Упражнение 1** Проверить установлена ли переменная OLDPWD<sup>18</sup>. Если установлена вывести сообщение ``your previous dir was '' и содержимое OLDPWD



---

<sup>18</sup> предыдущая рабочая директория, меняется 'cd'

## Практика: условия

**Упражнение 1** Проверить установлена ли переменная OLDPWD<sup>18</sup>. Если установлена вывести сообщение ``your previous dir was '' и содержимое OLDPWD

**Упражнение 2** Скрипт, проверяющий существование файла с публичными ключами SSH. И если он существует - вывести на экран



---

<sup>18</sup> предыдущая рабочая директория, меняется 'cd'

## Практика: условия

**Упражнение 1** Проверить установлена ли переменная OLDPWD<sup>18</sup>. Если установлена вывести сообщение ``your previous dir was '' и содержимое OLDPWD

**Упражнение 2** Скрипт, проверяющий существование файла с публичными ключами SSH. И если он существует - вывести на экран

**Упражнение 3** Скрипт, проверяющий наличие папки tmp. Если её нет - создать. После - сохранить в неё все переменные окружения (в любом виде)

---

<sup>18</sup> предыдущая рабочая директория, меняется 'cd'



## Циклы ``for'', ``while'', ``until''

for

```
for i in `list`  
do  
    commands  
done
```

В списке могут быть переменные, вызовы команд через обратные скобки, подстановочные символы (wildcards) и т.п.





## Циклы ``for'', ``while'', ``until''

for

```
for i in `list`  
do  
    commands  
done
```

В списке могут быть переменные, вызовы команд через обратные скобки, подстановочные символы (wildcards) и т.п.

while , until

```
while `list`  
do  
    commands  
done
```



## Практика: циклы

**Упражнение 1..** Скрипт: создать папку tmp и в ней файлы userX-номер, где X - ваш номер пользователя, номер - от 1 до 99. Файл - скопировать или создать командой touch.  
Получение чисел - команда seq



---

<sup>19</sup> для паузы можно использовать команду sleep

## Практика: циклы

**Упражнение 1..** Скрипт: создать папку tmp и в ней файлы userX-номер, где X - ваш номер пользователя, номер - от 1 до 99. Файл - скопировать или создать командой touch.

Получение чисел - команда seq

**Упражнение 2.** Скрипт: переименовать все полученные файлы из Упражнения 1 в ``ex-userX-номер''



---

<sup>19</sup> для паузы можно использовать команду sleep

## Практика: циклы

**Упражнение 1..** Скрипт: создать папку tmp и в ней файлы userX-номер, где X - ваш номер пользователя, номер - от 1 до 99. Файл - скопировать или создать командой touch.

Получение чисел - команда seq

**Упражнение 2.** Скрипт: переименовать все полученные файлы из Упражнения 1 в ``ex-userX-номер''

**Упражнение 3.** Скрипт: создать аналог ``ls -a'' с помощью циклов



---

<sup>19</sup> для паузы можно использовать команду sleep

## Практика: циклы

**Упражнение 1..** Скрипт: создать папку tmp и в ней файлы userX-номер, где X - ваш номер пользователя, номер - от 1 до 99. Файл - скопировать или создать командой touch.

Получение чисел - команда seq

**Упражнение 2.** Скрипт: переименовать все полученные файлы из Упражнения 1 в ``ex-userX-номер''

**Упражнение 3.** Скрипт: создать аналог ``ls -a'' с помощью циклов

**Упражнение 4.** Скрипт: генератор случайных чисел на экран. Раз в секунду<sup>19</sup> выводить случайное число на экран.



---

<sup>19</sup> для паузы можно использовать команду sleep

## IFS - internal field separator

### IFS

Переменная, регулирующая разделение параметров (аргументов) на слова.

Используется:

- во время раскрытия параметров командной строки перед выполнением
- редактирование командной строки (удаление слова, Ctrl+W)
- чтение ввода пользователя командной **read**

Значение по умолчанию: `` <пробел><табуляция><перевод строки>''



## Параметры скрипта

Сохраняются shell в специальных переменных окружения:

- **\$#** - количество параметров (аргументов) скрипта
- **\$1, \$2, ...\$9**<sup>20</sup> - параметры скрипта

```
./script aa bb # $1="aa" $2="bb" $# = 2  
./script "dd ee ff" # $1="dd ee ff" $# = 1  
./script gg\ hh # $1="gg hh" $# = 1
```

<sup>20</sup> По POSIX гарантирована поддержка 9 переменных

<sup>21</sup> Обрабатываются по-разному, в зависимости от значения в IFS



## Параметры скрипта

Сохраняются shell в специальных переменных окружения:

- **\$#** - количество параметров (аргументов) скрипта
- **\$1, \$2, ...\$9**<sup>20</sup> - параметры скрипта

```
./script aa bb # $1="aa" $2="bb" $# = 2  
./script "dd ee ff" # $1="dd ee ff" $# = 1  
./script gg\ hh # $1="gg hh" $# = 1
```

- **\$0** - имя самого скрипта
- **\$@, \$\***<sup>21</sup> - все параметры скрипта списком

<sup>20</sup> По POSIX гарантирована поддержка 9 переменных

<sup>21</sup> Обработываются по-разному, в зависимости от значения в IFS





## Параметры скрипта

Сохраняются shell в специальных переменных окружения:

- **\$#** - количество параметров (аргументов) скрипта
- **\$1, \$2, ...\$9**<sup>20</sup> - параметры скрипта

```
./script aa bb # $1="aa" $2="bb" $# = 2  
./script "dd ee ff" # $1="dd ee ff" $# = 1  
./script gg\ hh # $1="gg hh" $# = 1
```

- **\$0** - имя самого скрипта
- **\$@, \$\***<sup>21</sup> - все параметры скрипта списком

Встроенная команда **shift**: сдвигает параметры влево.

Уменьшит **\$#** на 1, перемещая **\$2** в **\$1**, **\$3** в **\$2** и т.д.

<sup>20</sup> По POSIX гарантирована поддержка 9 переменных

<sup>21</sup> Обработываются по-разному, в зависимости от значения в IFS



## Практика: параметры скрипта

**Упражнение 1** Скрипт, выводящий своё имя и параметры в обратном порядке.

Пример:

```
$ ./reverse_arg_script 1 jka 45 ыыы  
./reverse_arg_script ыы 45 jka 1
```



## Конфигурационные файлы

Многообразие конфигурационных файлов и их использования в разных Shell.

Все конфиги - скрипты на языке Shell, выполняются при старте интерпретатора и служат для дополнительной настройки окружения пользователя.

- `/etc/profile` глобальный profile<sup>22</sup>
- `$HOME/.profile` profile пользователя



---

<sup>22</sup>Bourne-совместимые шеллы: `bash(1)`, `ksh(1)`, `dash(1)`

## Конфигурационные файлы

Многообразие конфигурационных файлов и их использования в разных Shell.

Все конфиги - скрипты на языке Shell, выполняются при старте интерпретатора и служат для дополнительной настройки окружения пользователя.

- `/etc/profile` глобальный profile<sup>22</sup>
- `$HOME/.profile` profile пользователя
- См man на свой shell, раздел FILES.

**Внимание!** Поведение интерактивных и неинтерактивных (скрипты) запусков shell может отличаться набором запускаемых профайлов.



---

<sup>22</sup>Bourne-совместимые шеллы: `bash(1)`, `ksh(1)`, `dash(1)`

## Практика: конфигурационные файлы

**Упражнение 1** Включить цветное приглашение командной строки (color prompt) в `bash(1)`, отредактировав свой локальный профайл.

**Упражнение 2** Создать `alias` (псевдоним) `lsall` для команды `ls` в локальном профайле, добавив параметры рекурсивного просмотра всех файлов, включая скрытые.

**Упражнение 3** Добавить к пути поиска (`PATH`) произвольный каталог через редактирование локального профайла.

**Упражнение 4** Переопределить системную команду `editor` через `alias` в профайле, вписав туда тот редактор, которым вы обычно пользуетесь на занятиях.

**Упражнение 5** Добавить текущее время в приглашение командной строки (`PS1`)<sup>23</sup>

---

<sup>23</sup>Подсказка: раздел `PROMPTING` в мане на `bash`



## Арифметические вычисления

- `$((выражение))` - простая арифметика POSIX shell

```
num=8
```

```
echo $((num + 7))    # shows: 15
```

```
echo "3 + 4 = $((3+4))"  # shows: 3 + 4 = 7
```

```
: $((w = 2 + 3)); echo $w  # shows: 5
```

```
: $((x = w - 1)); echo $x  # shows: 4
```



## Арифметические вычисления

- **`$((выражение))`** - простая арифметика POSIX shell

```
num=8
```

```
echo $((num + 7))    # shows: 15
```

```
echo "3 + 4 = $((3+4))"  # shows: 3 + 4 = 7
```

```
: $((w = 2 + 3)); echo $w  # shows: 5
```

```
: $((x = w - 1)); echo $x  # shows: 4
```

- внешняя утилита **`bc`** - более сложные вычисления

Пример: синус от 1.4, вывод в hex, 4 знака после запятой

```
$ echo "obase=16; scale=4; s(1.4)"|bc -l  
.FC466EFA270812AC3
```

