

Занятие 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$ cd tmp
mend0za@ak112:/home/mend0za/tmp$ ls -la
..
...
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



¹Только у BASH и ZSH (если настроены)

Приёмы эффективной работы Автодополнение путей и команд - 1

Волшебная кнопка - TAB

- Имя команды



¹Только у BASH и ZSH (если настроены)

Приёмы эффективной работы Автодополнение путей и команд - 1

Волшебная кнопка - TAB

- Имя команды
Пример: `mys[TAB]_co[TAB]`

¹Только 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



¹Только 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

¹Только 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]`

¹Только 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

¹Только 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 ¹

¹Только 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 ¹
Пример: `apti[TAB]--a[TAB]sh[TAB]core[TAB][ENTER]`

¹Только 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 ¹
Пример: `apti[TAB]--a[TAB]sh[TAB]core[TAB][ENTER]`
Результат: `aptitude --assume-yes show coreutils`
16 vs 37

¹Только 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 ¹
Пример: `apti[TAB]--a[TAB]sh[TAB]core[TAB][ENTER]`
Результат: `aptitude --assume-yes show coreutils`
16 vs 37

¹Только 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:

ТАВ дополняет сразу

Несколько вариантов подстановки?

Ещё больше волшебства - 2 кнопки ТАВ!

2xТАВ - список вариантов подстановки



Приёмы эффективной работы Автодополнение путей и команд - 3

Примеры

- apt[TAB][TAB]
- aptitude --[TAB][TAB]²
- ls /[TAB][TAB]³

²Только для BASH и ZSH

³Можно 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 ⁴

- “Left” и “Right” - вперёд-назад по текущей строке

⁴Только KSH-совместимые: bash, zsh, pdksh, mksh, etc



Приёмы эффективной работы Редактирование командной строки

Emacs editing mode ⁴

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

⁴Только KSH-совместимые: bash, zsh, pdksh, mksh, etc



Приёмы эффективной работы Редактирование командной строки

Emacs editing mode ⁴

- “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

⁴Только KSH-совместимые: bash, zsh, pdksh, mksh, etc



Приёмы эффективной работы Редактирование командной строки

Emacs editing mode ⁴

- “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)

⁴Только 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 shell⁵
`./script`
`source script`⁶

⁵Остальные способы - запускают новый shell

⁶Несовместимо с POSIX. Происходит из ksh. Добавляет текущий каталог к списку путей



Потоки ввода-вывода

Особенности архитектуры⁷:

У каждой запущенной программы 3 потока I/O:

- 0 ввода
- 1 вывода
- 2 ошибок

Связаны с экраном и клавиатурой терминала.

⁷См документацию языка программирования Си



Потоки ввода-вывода

Особенности архитектуры⁷:

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

⁷См документацию языка программирования Си



Базовый синтаксис перенаправления

- Ввод “<”
sort <.bash_history



⁸Файл затрёт новым содержанием, если он существовал ранее

Базовый синтаксис перенаправления

- **Ввод** “<”
sort <.bash_history
- **Вывод** “>”⁸
find /usr/share/doc -name “*.txt” >txt-docs
- **Вывод** “1>”
find /usr/share/doc -name “*.txt” 1>txt-docs

⁸Файл затрёт новым содержанием, если он существовал ранее



Базовый синтаксис перенаправления

- **Ввод** “<”
sort <.bash_history
- **Вывод** “>”⁸
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

⁸Файл затрёт новым содержанием, если он существовал ранее



Базовый синтаксис перенаправления

- **Ввод** "<"
sort <.bash_history
- **Вывод** ">" 8
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

⁸Файл затрёт новым содержанием, если он существовал ранее



Базовый синтаксис перенаправления

- **Ввод** "<"
sort <.bash_history
- **Вывод** ">" 8
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

⁸Файл затрёт новым содержанием, если он существовал ранее



Расширенный синтаксис перенаправления

- **Pipe**⁹ “cmd1 | cmd2 ”

Вывод cmd1 направляется на ввод cmd2.

man bash|grep ksh



Расширенный синтаксис перенаправления

- **Pipe**⁹ “cmd1 | cmd2 ”

Вывод cmd1 направляется на ввод cmd2.

man bash|grep ksh

- **Склеить потоки** “N>&M”

В примере: просмотреть одновременно и вывод и ошибки

find /tmp 2>&1 | less



Расширенный синтаксис перенаправления

- **Pipe**⁹ “cmd1 | cmd2 ”

Вывод cmd1 направляется на ввод cmd2.

man bash|grep ksh

- **Склеить потоки** “N>&M”

В примере: просмотреть одновременно и вывод и ошибки

find /tmp 2>&1 | less

- **”Ввод здесь”**¹⁰ “<<END_MARKER”

```
sort <<EOF
```

```
oieu
```

```
ak
```

```
zf
```

```
EOF
```



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 скриптом¹¹.

¹¹ конструкция “ввод здесь”



Практика: перенаправление, скрипты - 2

Задание 3

Сделать вывод сохранённой в скрипте истории команд на экран.

Самим скриптом¹¹.

Задание 4

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

¹¹ конструкция “ввод здесь”



Практика: перенаправление, скрипты - 2

Задание 3

Сделать вывод сохранённой в скрипте истории команд на экран.

Самим скриптом¹¹.

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.

¹¹ конструкция “ввод здесь”



Переменные

Переменные:

настройки окружения пользователя для процесса ¹²



¹²Смотри `environ(7)` о подробностях реализации

Переменные

Переменные:

настройки окружения пользователя для процесса ¹²

Какие бывают?

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



¹²Смотри 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'
```



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"  
_____
```



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 -I{} sh -c "losetup -v {} \  
$FLASH_IMG"
```



Практика: переменные и скобки

Задание 1. Запустить скрипт с историей¹⁴ в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.

¹⁴из предыдущего задания по перенаправлению ввода-вывода



Практика: переменные и скобки

Задание 1. Запустить скрипт с историей¹⁴ в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.

Задание 2. Скрипт-генератор случайных чисел (RANDOM)

¹⁴из предыдущего задания по перенаправлению ввода-вывода



Практика: переменные и скобки

Задание 1. Запустить скрипт с историей¹⁴ 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.

- ❶ создать папку bin в домашнем каталоге
- ❷ дописать в PATH полный путь к созданной папке (используя переменную HOME)
- ❸ скопировать скрипт с историей в папку bin
- ❹ запустить скрипт, без указания пути к нему

¹⁴из предыдущего задания по перенаправлению ввода-вывода



Подстановочные символы путей (Wildcards)

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



¹⁵ об интервалах - в разделе о регулярных выражениях

Подстановочные символы путей (Wildcards)

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

- ***** - любое количество любых символов

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



¹⁵ об интервалах - в разделе о регулярных выражениях

Подстановочные символы путей (Wildcards)

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

- ***** - любое количество любых символов

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

- **[]** - символ из перечисления¹⁵

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

¹⁵ об интервалах - в разделе о регулярных выражениях



Подстановочные символы путей (Wildcards)

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

- ***** - любое количество любых символов

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

- **[]** - символ из перечисления¹⁵

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

- **?** - любой одиночный символ

```
echo ?i*
```



¹⁵ 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



¹⁶ 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"¹⁶, не используя "ls".



¹⁶ 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**¹⁷ числа равны
- **INTEGER1 -ne INTEGER2** числа не равны
- **INTEGER1 -gt INTEGER2** число 1 больше числа 2



¹⁷Привет учившим ассемблер

Практика: условия

Упражнение 1 Проверить установлена ли переменная OLDPWD¹⁸. Если установлена вывести сообщение “your previous dir was ” и содержимое OLDPWD



¹⁸предыдущая рабочая директория, меняется 'cd'

Практика: условия

Упражнение 1 Проверить установлена ли переменная OLDPWD¹⁸. 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

¹⁸предыдущая рабочая директория, меняется 'cd'



Практика: условия

Упражнение 1 Проверить установлена ли переменная OLDPWD¹⁸. 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)

¹⁸предыдущая рабочая директория, меняется 'cd'



Циклы “for” и “while”

for

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

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



Циклы “for” и “while”

for

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

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

while

```
while command or condition like test(1)
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



¹⁹ 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”



¹⁹ 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



¹⁹ 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 секунду¹⁹ выводить случайное число на экран.



¹⁹ для паузы можно использовать команду sleep