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/

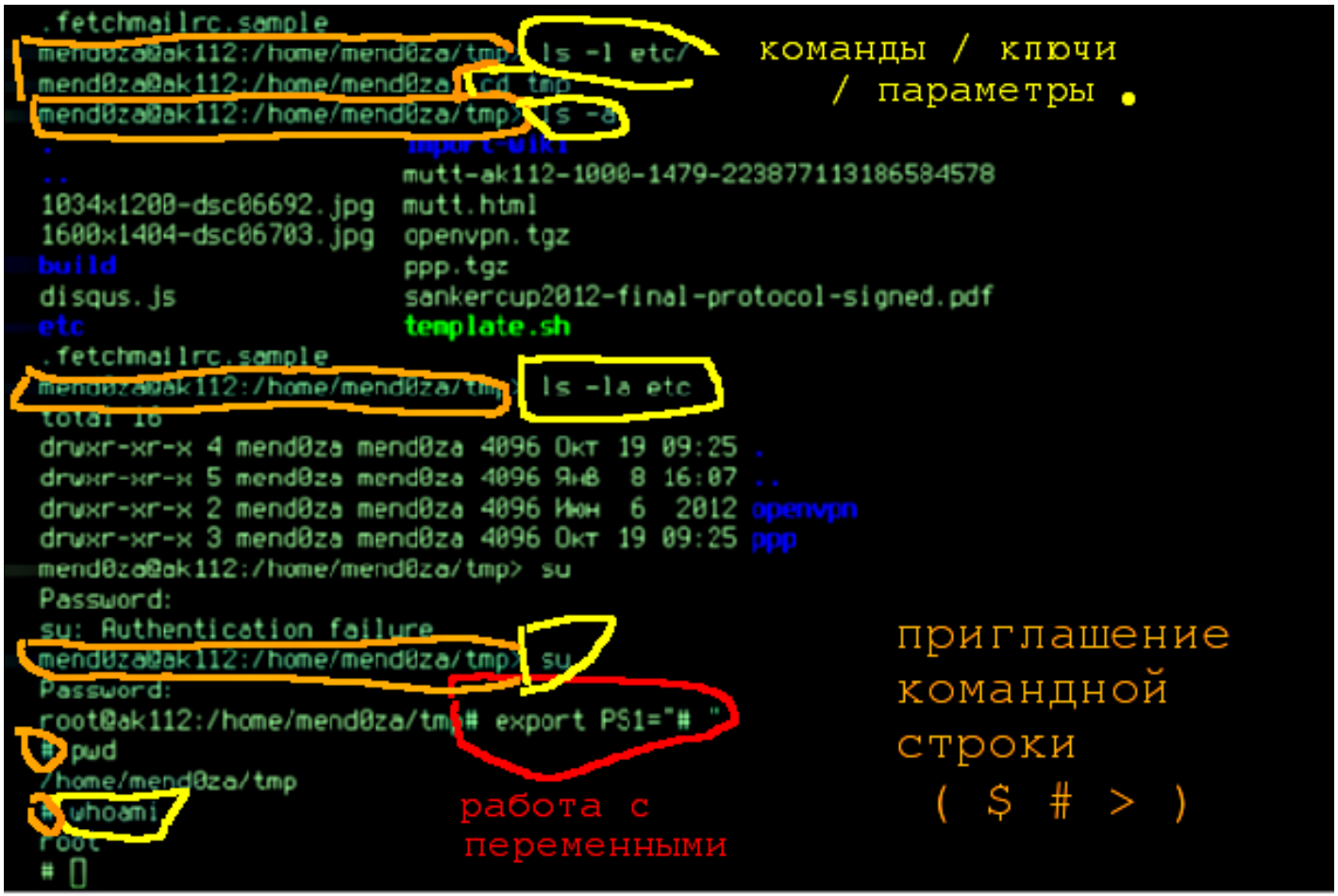
Shell. Ключевые понятия - 1

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



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

[1] Только у BASH и ZSH (если настроены)

Приёмы эффективной работы

Автодополнение путей и команд - 1

Волшебная кнопка - TAB

- Имя команды

[1] 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]

[1] 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

[1] 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

[1] 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]

[1] 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

[1] 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 [1]

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

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

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

[1] 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][2]
- ls /[TAB][TAB] [3]

[2] Только для BASH и ZSH
[3] 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 [4]

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

[4] Только 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

[4] Только 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

[4] Только 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)

[4] 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 - выполнень успешно
- не 0 - ошибка

Условное выполнение команд

Код возврата (RETURN CODE):

результат выполнения у любой команды Shell

Shell return code:

- 0 - выполнень 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 - выполнень 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

1. sh scriptname
2. chmod +x script
   ./script
3. из каталогов в переменной PATH
   echo $PATH
   ~/bin (если есть)
   /usr/local/bin
4. в текущей копии shell[5]
   . ./script
   source script[6]

[5] Остальные способы - запускают новый shell

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

Потоки ввода-вывода

Особенности архитектуры[7]:

У каждой запущенной программы 3 потока I/O:

0. ввода
1. вывода
2. ошибок

Связаны с экраном и клавиатурой терминала.

[7] См документацию языка программирования Си

Потоки ввода-вывода

Особенности архитектуры[7]:

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

[7] См документацию языка программирования Си

Базовый синтаксис перенаправления

- Ввод “<”

  sort <.bash_history

[8] Файл 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

[8] Файл 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

[8] Файл 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

[8] Файл 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

[8] Файл затрёт новым содержанием, если он существовал ранее

Расширенный синтаксис перенаправления

- Pipe [9] “cmd1 | cmd2 ”
  Вывод cmd1 направляется на ввод cmd2.
  man bash|grep ksh

[9] Классика Unix

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

[9] Классика Unix

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
```

[9] 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[11].

[11] конструкция “ввод здесь”

Практика: перенаправление, скрипты - 2

Задание 3

Сделать вывод сохранённой в скрипте истории команд на экран.

Самим скриптом[11].

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 скриптом[11].

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]

[12] Смотри 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

[12] 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'
```

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 кавычки

Задание 1. Запустить скрипт с историей[14] в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.

[14] из предыдущего задания по перенаправлению ввода-вывода

Практика: переменные и кавычки

Задание 1. Запустить скрипт с историей[14] в новом процессе, используя тот же SHELL, в котором вы работаете сейчас.
Задание 2. Скрипт-генератор случайных чисел (RANDOM)

[14] из предыдущего задания по перенаправлению ввода-вывода

Практика: переменные и кавычки

Задание 1. Запустить скрипт с историей[14] 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

[14] из 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 перечисления[15]

```
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 перечисления[15]

```
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'

[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'

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

[17] Привет учившим ассемблер

Практика: условия

Упражнение 1 Проверить установлена ли переменная OLDPWD[18]. Если установлена вывести сообщение “your previous dir was ” и содержимое OLDPWD

[18] передыдущая рабочая директория, меняется ’cd’

Практика: условия

Упражнение 1 Проверить установлена ли переменная OLDPWD[18]. 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

[18] передыдущая рабочая директория, меняется ’cd’

Практика: условия

Упражнение 1 Проверить установлена ли переменная OLDPWD[18]. 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)

[18] передыдущая рабочая директория, меняется ’cd’

Циклы “for”, “while”, “until”

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”, “while”, “until”

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 , until

```
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 секунду[19] выводить случайное число на экран.

[19] для паузы можно использовать команду sleep

Параметры скрипта

Сохраняются shell в специальных переменных окружения:

- $# - количество параметров (аргументов) скрипта
- $1, $2, ... $9[20] - параметры скрипта

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

[20] По POSIX гарантирована поддержка 9 переменных, в конкретных реализациях может быть больше

Параметры скрипта

Сохраняются shell в специальных переменных окружения:

- $# - количество параметров (аргументов) скрипта
- $1, $2, ... $9[20] - параметры скрипта

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

- $0 - имя самого скрипта
- $@ , $* - все параметры скрипта списком

[20] По POSIX гарантирована поддержка 9 переменных, в конкретных реализациях может быть больше