Архангельский государственный технический университет
Кафедра вычислительных систем и телекоммуникаций

Шайдо Павел Иванович

Командный интерпретатор Bash

Архангельск 2002

Содержание

Введение

Интерпретатор командного языка (или оболочка), это основное средство для взаимодействия пользователя с системой. Оболочка это не просто средство для запуска различных программ, но и довольно мощный язык программирования. Программы оболочки, называемые сценариями, позволяют легко объединять различные часто используемые вместе системные утилиты, для получения новых приложений.

Bash это реализация командного интерпретатора совместимая с интерпретатором sh, определенным в стандарте POSIX 1003.2, и содержащаяя, кроме того, некоторые дополнительные возможности.

1 Запуск интерпретатора

Интерпретатор может быть запущен в режиме сеансовой оболочки (командой login) или нет. При запуске интерпретатора в режиме сеансовой оболочки, он выполняет команды указанные в файле `/etc/profile`, если такой существует. Затем он ищет, файлы `~/.bash_profile`, `~/.bash_login` и `~/.profile` и выполняет первый существующий из них. При выходе из сеансовой оболочки выполняется файл `~/.bash_logout`.

Когда Bash запускается в интерактивном режиме, но не в качестве сеансовой оболочки, он выполняет файл `~/.bashrc`.

Когда Bash запускается не в интерактивном режиме, он извлекает имя файла, который содержит команды инициализации, из пременной окружения `BASH_ENV`.

2 Грамматические конструкции

2.1 Определения

слово — последовательность символов, воспринимаемая интерпретатором как одна единица.
имя — слово состоящее только из алфавитно-цифровых символов и знаков подчеркивания, начинающееся с буквы или знака подчеркивания. Также называется идентификатором.
метасимвол — символ, который, не будучи заключен в кавычки, разделяет слова. Один из следующих: `| & ; ( ) < > space tab`.
управляющий оператор — слово выполняющее функции управления. Один из следующих: `|| & && ; ;; ( ) | <newline>`.

2.2 Зарезервированные слова

Зарезервированные слова — это слова имеющие специальное значение для интерпретатора. Следующие слова являются зарезервированными:
! case do done elif else esac fi for function if in select then until while { } time [[ ]]

2.3 Простые команды

Простая команда это последовательность из необязательного присвоения значения переменной с последующими словами и перенаправлениями, прерываемая управляющим оператором. Первое слово определяет выполняемую команду. Последующие слова передаются команде в качестве аргументов.

Возвращаемое значение простой команды — код завершения или 128+n если команда была прервана по сигналу n.

2.4 Конвейеры

Конвейер — последовательность из одной или более команд, разделенных символом `|`. Формат конвейера следующий:

```
[time [-p]] [!] command [ | command2 ... ]
```

Стандартный вывод `command` подключается к стандартному вводу команды `command2`. Это подключение производится до выполнения любых перенаправлений.

Если конвейеру предшествует зарезервированное слово `!`, то код завершения конвейера равен логическому отрицанию кода завершения последней команды. Иначе код завершения конвейера равен коду завершения последней команды. Интерпретатор ожидает завершения всех команд до того как вернет значение.

Если конвейеру предшествует зарезервированное слово `time`, то после завершения выполнения конвейера будет выведена информация о времени выполнения конвейера и о затраченном времени процессора в режимах пользователя и системы.

Каждая команда в конвейере выполняется как отдельный процесс (т.е. в подоболочке).

2.5 Списки

Список — последовательность из одного или более конвейеров разделенных одним из операторов: `;, &, &&, ||`. Список может заканчиваться операторами `;, &, <newline>`. Среди этих операторов более высокий приоритет имеют `||` и `&&`, затем следуют операторы `|` и `&`.

Если команда заканчивается оператором `&`, то интерпретатор выолняет ее в фоновом режиме в подоболочке, при этом сразу возвращается код завершения ноль. Команды разделенные `;` выполняются последовательно, оболочка ожидает завершения каждой команды. Код завершения равен коду завершения последней выполненой команды.

Управляющие операторы `&&` и `||` выполняют логические операции И и ИЛИ. И списки имеют вид:

```
command1 && command2
```

Команда command2 выполняется тогда, и только тогда, когда код завершения команды command1 равен нулю.

Списки ИЛИ имеют вид:

```
command1 || command2
```

Команда command2 выполняется тогда, и только тогда, когда код завершения команды command1 не равен нулю. Код завершения списков И и ИЛИ равен коду завершения последней выполненой команды списка.

2.6 Составные команды

Составные команды могут быть следующими:
`(список)` — список выполняется в подоболочке. Присвоения переменных и встроенные команды, оказывающие воздействие на окружение интерпретатора, не будут иметь силы после завершения выполнения команды. Код завершения равен коду завершения списка.

`{ список; }` Список просто выполняется в текущей оболочке. Список должен прерываться символом завершения строки или точкой с запятой. Такая команда называется групповой. Код завершения равен коду завершения списка.

`(( выражение ))` Вычисляет значение выражения. Если значение выражения не равно нулю, то код завершения ноль, в противном случае код завершения равен единице.

`[[ выражение ]]` Вычисляет значение условного выражения. Если значение истина, то код завершения ноль, в противном случае единица.

Выражения могут объединяться следующими операторами, перечисленными в порядке убывания приоритета:

- `( выражение )` — возвращает значение выражения.
- `! выражение` — истина если выражение ложь.
- `выражение1 && выражение2` — истина если оба выражения истина.
- `выражение1 || выражение2` — истина если хотя бы одно из выражений истина.

Операторы `&&` и `||` не выполняют выражение2 если значения выражения1 достаточно для определения результата.

`for имя [ in слово ] ; do список ; done`
Список слов, следующих за `in` раскрывается, образуя последовательность значений. Переменной `имя` поочередно присваиваются все эти значения, при этом каждый раз выполняется `список`. Если конструкция `in слово` опущена, список выполняется один раз для каждого установленного позиционного параметра. Код завершения равен коду завершения последней выполненой команды. Если раскрытие элементов следующих за `in` дает в результате пустой список, то выполнения команд не происходит и код завершения равен нулю.

`for (( выраж1 ; выраж2 ; выраж3 )) ; do список ; done`
Сначала вычисляется арифметическое выражение `выраж1`. Затем в цикле вычисляется значение выражения `выраж2`, если оно не равно нулю, то выполняется список и вычисляется значение `выраж3`. Цикл повторяется пока значение `выраж2` не станет равным нулю. Если какое-либо выражение опущено, то его значение предполагается равным единице.

`select имя [ in слово ] ; do список ; done`
Список слов, следующих за `in` раскрывается, образуя последовательность значений. Множество получнных значений выводится на стандартный вывод ошибок, каждое значение предваряется номером. Если конструкция `in слово` опущена, печатается список позиционных параметров. Затем выводится промпт PS3 и считывается строка из стандартного ввода. Если строка содержит номер соответствующий одному из выведенных слов, то значение переменной `имя` устанавливается равным этому слову. Если строка пустая, то список печатается повторно. Если строка содержит EOF, то команда завершается. Любое другое значение приводит к тому, что переменная `имя` устанавливается в ноль. Полученная строка сохраняется в переменной REPLY. После каждого выбора выполняется `список`. Цикл повторяется до выполнения команды `break` или `return`. Код завершения равен коду завершения последней выполненой команды или нулю если команды не выполнялись.

```
case слово in
    [ ( шаблон [ | шаблон ] ... ) список ;; ] ...
esac
```

Команда сначала раскрывает `слово` и ставит его в соответствие каждому `шаблону`. Когда совпадение найдено выполняется соответствующий `список`. После первого совпадения попытки

найти соответствующий шаблон прекращаются. Код завершения равен нулю если найти совпадение не удалось и коду завершения последней выполненой команды в противном случае.

```
if список1; then список2;
[ elif список3; then сисок4; ] ...
[ else список5; ] fi
```

Выполняется `список1` если его код завершения ноль, то выполняется `список2`. Иначе выполняются поочередно все конструкции `elif`. Если `список3` завершается с кодом ноль, то выполняется `список4` и команда завершается. В противном случае выполняется `список5`. Код завершения равен коду завершения последней команды или нулю если ни одно из условий не было выполнено.

```
while список1; do список2; done
until список1; do список2; done
```

Команда while выполняет список2 пока список1 завершается с кодом ноль. Команда until аналогична, но выполняет список2 до тех пор пока список1 не завершится с кодом ноль. Код завершения команды равен коду завершения последней выполненой команды или нулю если ни одной команды не было выполнено.

```
[ function ] имя () { список; }
```

Команда определяет функцию с именем **имя**. Тело функции содержится в фигурных скобках. Этот список выполняется когда **имя** используется в качестве простой команды. Код завершения функции равен коду завершения последней выполненой в теле функции команды.

3 Комментарии

В неинтерактивном режиме оболочка воспринимает слова начинающиеся с # и все остальные символы до конца строки как комментарии и игнорирует их. В интерактивном режиме комментарии как правило не допускаются.

4 Кавычки

Заключение в кавычки (цитирование) используется для того, чтобы отключить специальное значение некоторых символов для оболочки. Кавычки отменяют специальную трактовку некоторых символов, распознавание зарезервированных слов, раскрытие параметров.

Любой метасимвол имеет специальное значение для оболочки и должен заключаться в кавычки чтобы обозначать себя. Для этого имеется три механизма: escape символ, одинарные кавычки, двойные кавычки.

Бэкслэш, не будучи заключен в кавычки, является escape символом. Он сохраняет значение символа следующего за ним, за исключением символа новой строки. Последовательность `\<newline>` трактуется как продолжение строки (т.е. она удаляется из входного потока и полностью игнорируется).

Заключение строки символов в одинарные кавычки сохраняет значение каждого символа строки. Одинарная кавычка не может быть в этой строке, даже если ей предшествует бэкслэш.

Заключение символов в двойные кавычки сохраняет значение каждого символа в строке за исключением `$, ', \`. Символы `$` и `'` сохраняют свое спецальное значение внутри двойных кавычек. Бэкслэш сохраняет свое специальное значение только если предшествует одному из следующих символов: `$, ', ", \, <newline>`.

Специальные параметры `*` и `@` имеют особое значение в двойных кавычках.

Особым образом трактуются слова вида `$'string'`. Такие слова преобразуются в строки с заменой отделенных бэкслэшем символов в соответствии со стандартом языка C:

`\a`	звуковой сигнал
`\b`	backspace
`\e`	escape
`\f`	прогон страницы
`\n`	новая строка
`\r`	возврат каретки
`\t`	горизонтальная табуляция
`\v`	вертикальная табуляция
`\\`	бэкслэш
`\nnn`	символ с восьмеричным кодом nnn
`\xnnn`	символ с шестнадцатеричным кодом nnn

5 Параметры

Параметр — это объект хранящий значение. Параметр установлен если ему присвоено значение. Null является допустимым значением. Если параметр установлен, то он может перейти в состояние "не установлен" только при использовании встроенной команды `unset`. Параметр может быть обозначен именем, числом или специальным символом. *Переменная* — это параметр обозначенный именем.

5.1 Позиционные параметры

Позиционный параметр — это параметр обозначенный одной или более цифрами, отличными от единственной цифры 0. Если параметр обозначен несколькими цифрами, они должны заключаться в фигурные скобки. Позиционные параметры определяются аргументами указанными в командной строке при вызове оболочки. Позиционные параметры временно заменяются при вызове функций. Присвоение значений позиционным параметрам может быть произведено при помощи встроенной команды `set`.

5.2 Специальные параметры

Некоторые параметры трактуются оболочкой особым образом. Значение этих параметров можно получить, но изменить его нельзя.

`*` Раскрывается в строку позиционных параметров начиная с первого. Если используется внутри двойных кавычек, то преобразуется в одно слово, состоящее из позиционных параметров разделенных первым символом переменной `IFS`. Если IFS null, то параметры объединяются, если IFS не установлена, то параметры разделяются пробелами.

`@` Раскрывается в строку позиционных параметров начиная с первого. Если используется внутри двойных кавычек, то каждый позиционный параметр преобразуется в отдельное слово.

`#` Число позиционных параметров в десятичном виде.

`?` Статус завершения последнего выполненого конвейера.

`-` Список опций заданных в командной строке, установленных встроенной командой `set` или самой оболочкой.

`$` ID процесса оболочки. В подоболочках `$` сохраняет значение и выдает ID процесса оболочки, а не подоболочки.

`!` ID процесса последней выполненной в фоновом режиме команды.

`0` Имя оболочки или сценария.

5.3 Переменные оболочки

Присвоение значения переменной выполняется записью вида:

```
name=[value]
```

Если значение не задано, переменной присваивается значение null. Все значения подвергаются процедурам раскрытия тильд, параметров, переменных и строк, подстановки команд, удаления кавычек (см. далее).

Имеются следующие стандартные переменные.

ENV Переменная содержит имя файла, который будет выполнен при запуске оболочки.

HOME Путевое имя домашнего каталога пользователя. Используется при замене тильды.

IFS Список символов, используемых для разделения слов во входном потоке.

LANG Значение по умолчанию для переменных локализации, которые не установлены или имеют значение null.

LC_ALL
Определяет значение всех переменных локализации.

LC_COLLATE
Данная переменная определяет порядок сортировки и вычисление значений промежутков.

LC_CTYPE
Переменная определяет значения классов символов.

LC_MESSAGES
Переменная определяет язык на котором будут выводиться сообщения.

LINENO
При выполнении сценариев переменная равна номеру строки файла.

NLSPATH
Строка, содержащая пути для поиска команд. Является списком каталогов разделенных двоеточием.

PPID Содержит идентификатор родительского процесса. В подоболочках не изменяется.

PS1 Каждый раз, когда интерактивная оболочка готова принять очередную команду, значение переменной раскрывается и выводится в стандартный поток ошибок.

PS2 Каждый раз, когда пользователь вводит символ новой строки до завершения команды, интерактивная оболочка выводит значение переменной в стандартный поток ошибок. По умолчанию равна >.

PS3 Значение этой переменной используется в качестве подсказки для команды select.

PS4 Используется при трассировке команд.

6 Раскрытие слов

Раскрытие производится после разделения командной строки на слова. Имеется несколько типов раскрытия: раскрытие тильды, раскрытие параметров, подстановка команд, арифметические преобразования, разделение слов, подстановка путевых имен и удаление кавычек. При разделении слов и подстановке путевых имен число слов может изменяться, остальные преобразования оставляют число слов неизменным.

6.1 Раскрытие тильды

Если слово начинается с незаключенной в кавычки тильды, то все символы предшествующие первому слэшу (или все символы, если слэша нет) составляют тильд-префикс. Если строка символов префикса после тильды не содержит символов заключенных в кавычки, то она трактуется как возможное имя пользователя. Если это имя пользователя — нулевая строка, то тильда заменяется значением переменной *HOME* или домашним каталогом пользователя выполняющего команду. Иначе тильд-префикс заменяется домашним каталогом указанного пользователя. Если пользователя с указанным именем нет, то результат не определен.

6.2 Раскрытие параметра

Символ `$` в слове приводит к раскрытию параметра, подстановке команды или вычислению арифметического выражения. Формат параметра следующий:
`${выражение}`

В простейшем случае выражение является именем параметра. В этом случае подставляется значение параметра. Если фигурные скобки опущены, то используется самое длинное допустимое имя, независимо от того определена ли переменная с таким именем.

Кроме того существуют дополнительные форматы раскрытия параметра. В этих случаях *word* подвергается раскрытию тильды, раскрытию параметра, подстановке команды и вычислению арифметического значения:

${parameter:-word}

Использовать значения по умолчанию. Если параметр не установлен или null, подставляется *word*; иначе подставляется значение параметра.

${parameter:=word}

Присвоить значения по умолчанию. Если параметр не установлен или null, то ему присваивается значение *word*. В любом случае, в конце подставляется значение параметра. Так могут использоваться только переменные, но не позиционные параметры и не специальные параметры.

${parameter:?[word]}

Выдать ошибку если параметр null или не установлен. Если параметр имеет значение null или не установлен, то *word* выводится в стандартный поток ошибок и выполнение интерпретатора завершается с ненулевым кодом возврата. В противном случае возвращается значение параметра. Выполнение интерактивной оболочки не прерывается.

${parameter:+word}

Использовать альтернативное значение. Если параметр не установлен или null, то ничего не подставляется. В противном случае подставляется значение *word*.

${#parameter}

Длина строки. Количество символов в значении параметра. Если параметр `*` или `@`, результат не определен.

${parameter%word}

Удалить наименьший суффикс. Значение *word* воспринимается как шаблон. Результатом является значение параметра с удаленным наименьшим суффиксом соответствующим шаблону.

${parameter%%word}

Удалить наибольший суффикс. Значение *word* воспринимается как шаблон. Результатом является значение параметра с удаленным наибольшим суффиксом соответствующим шаблону.

${parameter#word}

Удалить наименьший префикс. Значение *word* воспринимается как шаблон. Результатом является значение параметра с удаленным наименьшим префиксом соответствующим шаблону.

${parameter##word}

Удалить наибольший префикс. Значение *word* воспринимается как шаблон. Результатом является значение параметра с удаленным наибольшим префиксом соответствующим шаблону.

6.3 Подстановка команд

Подстановка команд позволяет заменять вывод имя команды ее выводом. Подстановка осуществляется, когда имя команды записано в форме:

```
$(command)
```

или

```
`command`
```
[1]

Оболочка выполняет команду и заменяет выражение выводом команды. Последовательность символов новой строки в конце вывода удаляется. Промежуточные символы новой строки не удаляются, обычно они трактуются как разделители слов и убираются при разделении слов.

6.4 Арифметические вычисления

Оболочка позволяет производить арифметические вычисления и подставлять результат вместо арифметических выражений. Для этого выражения записываются в виде:

```
$((expression))
```

[1] Обратите внимание, что здесь используются **обратные** одинарные кавычки

Выражение рассматривается как строка заключенная в двойные кавычки. Интерпретатор производит над выражением операции раскрытия параметров, подстановки команд и удаления кавычек. Вычисления производятся согласно правилам описаным ниже.

6.5 Разделение на слова

После раскрытия параметров, подстановки команд и вычисления арифметических выражений оболочка разделяет полученные результаты, не заключенные в двойные кавычки, на слова. В качестве разделителей используются символы переменной IFS. Если эта переменная не определена, то используются символы пробела, табуляции и новой строки. Если переменная IFS пустая, то разделение на слова не производится.

6.6 Раскрытие путевых имен

После разделения на слова оболочка проверяет каждое слово на наличие символов `* ? [`. Если какой-либо из этих символов присутствует, то слово трактуется как шаблон и заменяется именами файлов соответствующих данному шаблону. Символ `?` соответствует любому символу. Символ `*` соответствует любой строке, в том числе пустой. Символ `[` открывает выражение в квадратных скобках.

Выражение в квадратных скобках соответствует любому из заключенных в него символов. Если первым символом после [является `!` или `^`, то выражение в скобках соответствует любому символу не входящему в него. Возможно задание диапазонов символов, напр. `A-Z`. Кроме того, допускается указание классов символов, напр. `[[:punct:]]`. Определены следующие классы: alnum, alpha, ascii, blank, cntrl, digit, graph, lower, print, punct, space, upper, xdigit.

Примеры:

```
a?c
a*c
a[bdf]c
a[[:lower:]]c
a[![:punct:]]c
```

Файл с именем abc будет удовлетворять всем этим выражениям.

6.7 Удаление кавычек

После выполнения всех раскрытий и подстановок, символы `\ ' "`, присутствовавшие в исходной строке и не заключенные в кавычки, удаляются.

7 Перенаправление

До запуска команды ее ввод и вывод могут быть перенаправлены. Операторы перенаправления могут использоваться в любом месте простой команды. Обработка операторов перенаправления осуществляется слева направо, порядок следования операторов имеет значение.

В нижеследующем описании, в случаях когда дескриптор файла опущен, а оператор перенаправления — “<”, подразумевается стандартный ввод (дескриптор файла 0), если оператор перенаправления — “>”, то подразумевается стандартный вывод (дескриптор файла 1).

7.1 Перенаправление ввода

Для перенаправления ввода используется выражение вида:

```
[n]<слово
```

Слово подвергается операциям раскрытия, открывается для чтения файл с полученным в результате именем на указанном дескрипторе `n`, или на стандартном вводе, если `n` не задан.

7.2 Перенаправление вывода

Для перенаправления вывода используется выражение вида:

```
[n]>слово
```

Слово подвергается операциям раскрытия, открывается для записи файл с полученным в результате именем на указанном дескрипторе `n`, или на стандартном выводе, если `n` не задан. Если файл ранее существовал, то он усекается до нулевой длины, если файла с указанным именем не существует, то он создается.

Если использовать для перенаправления вывода оператор вида:

```
[n]>>слово
```

то файл, если он уже существует, будет открыт для добавления. Выводимые данные будут дописаны в конец файла.

7.3 Внедренные документы

Данный тип перенаправления считывать ввод из текущего источника команд. Например, если перенаправление используется в сценарии, то данные будут считываться из файла сценария начиная с оператора перенаправления и до строки содержащей только `слово`. Формат внедренных документов следующий:

```
<<слово
  данные
разделитель
```

Слово не подвергается операциям раскрытия. Если какой-либо символ в `слове` заключен в кавычки, то `разделитель` является результатом удаления кавычек, а данные не подвергаются операциям раскрытия. Если `слово` не содержит символов заключенных в кавычки, то данные подвергаются операциям раскрытия параметров, подстановки команд и раскрытию арифметических выражений.

Если оператор перенаправления завершается знаком минус – “`<<-`”, то все лидирующие символы табуляции удаляются из строк данных.

7.4 Дублирование дескрипторов файлов

Операторы:

```
[n]<&слово
[n]>&слово
```

Используются для дублирования дескрипторов файлов. Если слово раскрывается в число, и дескриптор файла определяемый данным числом открыт на чтение (запись), то этот дескриптор копируется в дескриптор `n`.

7.5 Перенаправление для чтения и записи

Оператор перенаправления вида:

`[n]<>слово`

приводит к открытию файла, определенного словом, для чтения и записи на дескрипторе `n`. Если `n` опущен, то он предполагается равным нулю.

8 Арифметические выражения

Оболочка позволяет вычислять арифметические выражения. Вычисления производятся с длинными целыми, без проверки переполнения, деление на ноль приводит к ошибке. Ниже перечислены арифметические операторы в порядке убывания приоритета.

- + унарные плюс и минус

! ~ логическое и побитовое отрицание

****** возведение в степень

*** / %** умножение, деление, остаток

+ - сложение, вычитание

< < > >
сдвиг битов

<= >= < >
сравнение

== != равенство, неравенство

& побитовое И

^ побитовое исключающее ИЛИ

| побитовое ИЛИ

&& логическое И

|| логическое ИЛИ

expr1?expr2:expr3
условное выражение

= *= /= %= += -= < <= > >= &= ^ = |=
присвоение

В качестве операндов могут использоваться переменные. До вычисления выражения производится раскрытие параметров. Значение параметра приводится к длинному целому.

Константы начинающиеся с 0 интерпретируются как восьмеричные числа, начинающиеся с 0x или 0X как шестнадцатеричные. Иначе числа записываются в форме [base#]n, где base — число от 2 до 64 определяющее систему счисления, а n число в этой системе. Если base опущено, то число считается десятичным. Цифры больше 9 представляются латинскими буквами в нижнем регистре, верхнем регистре, _, @, в данном порядке. Если основание не превышает 36, то большие и маленькие буквы равнозначны.

9 Условные выражения

Условные выражения используются в составной команде [[ ]] и во встроенных командах test и [ ]. Выражения формируются из следующих примитивов:

-a file Истина если файл существует

-b file Истина если файл существует и это файл блочно-ориентированного устройства

-c file Истина если файл существует и это файл байт-ориентированного устройства

-d file Истина если файл существует и это каталог

-e file Истина если файл существует

-f file Истина если файл существует и это регулярный файл

-g file Истина если файл существует и у него установлен бит SGID

-h file Истина если файл существует и это символическая ссылка

-k file Истина если файл существует и у него установлен бит SUID

-p file Истина если файл существует и это именованный канал

-r file Истина если файл существует и он доступен для чтения

-s file Истина если файл существует и его размер больше нуля

-t fd Истина если файл с указанным дескриптором открыт и это терминал

-u file Истина если файл существует и у него установлен бит SUID

-w file Истина если файл существует и доступен для записи

-x file Истина если файл существует и этот файл выполняемый

-O file Истина если файл существует и его владелец — пользователь чей идентификатор равен эффективному идентификатору выполняемого процесса

-G file Истина если файл существует и принадлежит группе идентификатор которой равен эффективному идентификатору группы выполняемого процесса

-L file Истина если файл существует и это символическая ссылка

-S file Истина если файл существует и это сокет

-N file Истина если файл существует и время изменения больше времени доступа

file1 -nt file2
Истина если file1 новее file2

file1 -ot file2
Истина если file1 старее file2

-z string
Истина если длина строки ноль

-n string
Истина если длина строки не ноль

string1 == string2
: Истина если строки равны

string1 != string2
: Истина если строки не равны

string1 < string2
: Истина если string1 при сортировке, с учетом текущей локали, окажется раньше string2

string1 > string2
: Истина если string1 при сортировке окажется после string2

arg1 OP arg2
: , где OP одна из следующих: -eq, -ne, -lt, -le, -gt, -ge. Истина в случае выполнения соответствующего условия.

10 Встроенные команды оболочки

. filename [arguments]
: Читает и выполняет команды из указанного файла в текущем процессе.

alias [name = value]
: Без аргументов печатает список существующих алиасов. С аргументами определяет новый алиас.

break [n]
: Прерывает выполнение цикла while, until, for или select. n определяет уровень циклов, которые будут прерваны.

cd [dir] Смена текущего каталога. Без аргументов меняет каталог на домашний. Если вместо каталога стоит минус, меняет каталог на предыдущий.

continue [n]
: Заканчивает текущую итерацию цикла. n определяет уровень цикла.

echo [-ne][arguments]
: Выводит аргументы, разделенные пробелами, с последующим символом новой строки. Код завершения всегда равен нулю. Если используется с ключом -n, то опускается завершающий символ новой строки. Если используется -e, то включается интерпретация следующих за бэкслешем символов:

```
\a \b \f \n \t \v \\ \nnn \xnnn
```

exec [-c][-a name]command [arguments]
: Команда command замещает текущую оболочку. Не порождается новый процесс. -c приводит к очистке окружения. -a определяет нулевой аргумент запускаемой команды.

exit [n] Приводит к завершению выполнения оболочки с кодом завершения n. Если n опущена, то код завершения равен коду завершения последней выполненой команды.

export [-p][name[= value]]
: Переменная с указанным именем помечается как экспортируемая, т.е. она становится доступной в порождаемых процессах. Если указан ключ -p, то печатается список всех экспортируемых переменных.

kill [-s signame | -signame]pid
: Посылает сигнал signame процессу с идентификатором pid. signame может быть именем сигнала или его номером. Если используется имя, то префикс sig может быть опущен. Основные сигналы: sigkill, sighup, sigterm, sigstop, sigcont, sigusr1, sigusr2.

read [-r]var ...

Считывает строку со стандартного ввода. Если не указан ключ -r, то бэкслэш является специальным символом. Считанные слова присваиваются переменным указанным в качестве аргументов.

readonly [-p][name[=value]]

Переменные, чьи имена указаны среди аргументов помечаются только для чтения. При использовании ключа -p выводится список переменных помеченных только для чтения.

return [n]

Приводит к завершению выполнения функции или файла. Параметр n определяет код завершения.

set [{+|-}abCef mnuvx][argument ...]

При запуске без аргументов интерпретатор выводит имена и значения всех переменных оболочки. Если указаны аргументы, то они определяют значения позиционных параметров. Ключи имеют следующие значения:

-a Все создаваемые или изменяемые переменные будут помечаться как экспортируемые.

-b Отчет о завершении фонового процесса будет выдаваться немедленно после его завершения.

-C Предотвращает перезапись файлов при использовании оператора перенаправления >.

-e Если простая команда завершится с ненулевым кодом, оболочка завершит свое выполнение.

-f Отменить раскрытие путевых имен.

-m Мониторинг фоновых заданий.

-n Считывать задания, но не выполнять их. Используется для проверки синтаксиса. Игнорируется интерактивными оболочками.

-u При раскрытии несуществующих переменных интерпретатор будет выдавать ошибку.

-v Оболочка будет копировать весь стандартный ввод в стандартный поток ошибок.

-x Трассировка команд.

shift [n]

Сдвиг позиционных параметров. Позиционному параметру 1 будет присвоено значение позиционного параметра 1+n, и т.д.

times Выводит затраченное время процессора в режимах пользователя и системы на выполнение интерпретатора и порожденных им дочерних процессов.

trap [action condition ...]

Устанавливает ловушку на событие. Если action равно -, то происходит сброс ловушек для события. Событие может быть EXIT или имя сигнала.

unset name ...

Уничтожает переменную.

wait [n]

Ожидает завершения указанного процесса и возвращает код его завершения. Если n опущена, то ожидает завершения всех дочерних процессов и возвращает код завершения ноль.

Источники

1. Bash Reference Manual. Brian Fox, Chet Ramey.
2. The Single UNIX (R) Specification, Versin 2. The Open Group. 1997. http://www.unix-systems.org
3. Advanced Bash-Scripting Guide: A complete guide to shell scripting, using Bash. Mendel Cooper. 2001. http://personal.riverusers.com/ thegrendel/