Front matter

title: "Лабораторная работа № 10" subtitle: "Программирование в командномпроцессоре ОС UNIX. Командные файлы" author: "Pavlova Polina"

Generic otions

lang: ru-RU toc-title: "Содержание"

Bibliography

bibliography: bib/cite.bib csl: pandoc/csl/gost-r-7-0-5-2008-numeric.csl

Pdf output format

toc: true # Table of contents toc-depth: 2 lof: true # List of figures lot: true # List of tables fontsize: 12pt linestretch: 1.5 papersize: a4 documentclass: scrreprt

I18n polyglossia

polyglossia-lang: name: russian options: - spelling=modern - babelshorthands=true polyglossia-otherlangs: name: english

I18n babel

babel-lang: russian babel-otherlangs: english

Fonts

mainfont: PT Serif romanfont: PT Serif sansfont: PT Sans monofont: PT Mono mainfontoptions: Ligatures=TeX romanfontoptions: Ligatures=TeX sansfontoptions: Ligatures=TeX,Scale=MatchLowercase monofontoptions: Scale=MatchLowercase,Scale=0.9

Biblatex

biblatex: true biblio-style: "gost-numeric" biblatexoptions: - parentracker=true - backend=biber - hyperref=auto - language=auto - autolang=other* - citestyle=gost-numeric

Pandoc-crossref LaTeX customization

figureTitle: "Рис." tableTitle: "Таблица" listingTitle: "Листинг" lofTitle: "Список иллюстраций" lotTitle: "Список таблиц" lolTitle: "Листинги"

Misc options

indent: true header-includes: - \usepackage{indentfirst} - \usepackage{float} # keep figures where there are in the text

- \floatplacement{figure}{H} # keep figures where there are in the text

Лабораторная работа № 10

Программирование в командном процессоре ОС UNIX. Командные файлы

Pavlova Polina

Цель работы

Изучить основы программирования в оболочке ОС UNIX/Linux. Научиться писать небольшие командные файлы.

Задание

1. Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в

другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.

2. Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов.
3. Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.
4. Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.

Теоретическое введение

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов. Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению.

Выполнение лабораторной работы

Написать скрипт, который при запуске будет делать резервную копию самого себя (то есть файла, в котором содержится его исходный код) в другую директорию backup в вашем домашнем каталоге. При этом файл должен архивироваться одним из архиваторов на выбор zip, bzip2 или tar. Способ использования команд архивации необходимо узнать, изучив справку.(рис.1.1-1.4)

image

Рис.1.1 Создание файла

image

Рис.1.2 Написание кода

image

Рис.1.3 Выдача прав и выполнение файла

image

Рис.1.4 Проверка файла на исполнение

Написать пример командного файла, обрабатывающего любое произвольное число аргументов командной строки, в том числе превышающее десять. Например, скрипт может последовательно распечатывать значения всех переданных аргументов. (рис.1.5-1.7)

image

Рис.1.5 Создание файла

image

Рис.1.6 Написание кода

image

Рис.1.7 Выдача прав и выполнение файла

Написать командный файл — аналог команды ls (без использования самой этой команды и команды dir). Требуется, чтобы он выдавал информацию о нужном каталоге и выводил информацию о возможностях доступа к файлам этого каталога.(рис.1.8-1.10)

image

Рис.1.8 Создание файла

image

Рис.1.9 Написание кода

image

Рис.1.3 Выдача прав и выполнение файла

image

Рис.1.10 Проверка файла на исполнение

Написать командный файл, который получает в качестве аргумента командной строки формат файла (.txt, .doc, .jpg, .pdf и т.д.) и вычисляет количество таких файлов в указанной директории. Путь к директории также передаётся в виде аргумента командной строки.(рис.1.11-1.14)

image

Рис.1.11 Создание файла

image

Рис.1.12 Написание кода

image

Рис.1.13 Выдача прав и выполнение файла

image

Рис.1.14 Проверка файла на исполнение

Выводы

Получен навык написания небольших команд.

Ответы на контрольные вопросы

1. Объясните понятие командной оболочки. Приведите примеры командных оболочек. Чем они отличаются?

Командный процессор (командная оболочка, интерпретатор команд shell) — это программа, позволяющая пользователю взаимодействовать с операционной системой компьютера. В операционных системах типа UNIX/Linux наиболее часто используются следующие реализации командных оболочек: – оболочка Борна (Bourne shell или sh) — стандартная командная оболочка UNIX/Linux, содержащая базовый, но при этом полный набор функций; – C-оболочка (или csh) — надстройка на оболочкой Борна, использующая C-подобный синтаксис команд с возможностью сохранения истории выполнения команд; – оболочка Корна (или ksh) — напоминает оболочку C, но операторы управления программой совместимы с операторами оболочки Борна; – BASH — сокращение от Bourne Again Shell (опять оболочка Борна), в основе своей совмещает свойства оболочек C и Корна (разработка компании Free Software Foundation).

2. Что такое POSIX?

POSIX (Portable Operating System Interface for Computer Environments) — набор стандартов описания интерфейсов взаимодействия операционной системы и прикладных программ.

3. Как определяются переменные и массивы в языке программирования bash?

Командный процессор bash обеспечивает возможность использования переменных типа строка символов. Имена переменных могут быть выбраны пользователем. Пользователь имеет возможность присвоить переменной значение некоторой строки символов.

4. Каково назначение операторов let и read?

Оболочка bash поддерживает встроенные арифметические функции. Команда let является показателем того, что последующие аргументы представляют собой выражение, подлежащее вычислению. Простейшее выражение — это единичный терм (term), обычно целочисленный. Команда read позволяет читать значения переменных со стандартного ввода.

5. Какие арифметические операции можно применять в языке программирования bash?

! !exp Если exp равно 0, то возвращает 1; иначе 0 != exp1 !=exp2 Если exp1 не равно exp2, то возвращает 1; иначе 0 % exp1%exp2 Возвращает остаток от деления exp1 на exp2 %= var=%exp Присваивает остаток от деления var на exp переменной var & exp1&exp2 Возвращает побитовое AND выражений exp1 и exp2 && exp1&&exp2 Если и exp1 и exp2 не равны нулю, то возвращает 1; иначе 0 &= var &= exp Присваивает переменной var побитовое AND var и exp * exp1 * exp2 Умножает exp1 на exp2 *= var *= exp Умножает exp на значение переменной var и присваивает результат переменной var + exp1 + exp2 Складывает exp1 и exp2 += var += exp Складывает exp со значением переменной var и результат присваивает переменной var - -exp Операция отрицания exp (унарный минус) - expl - exp2 Вычитает exp2 из exp1 -= var -= exp Вычитает exp из значения переменной var и присваивает результат переменной var / exp / exp2 Делит exp1 на exp2 /= var /= exp Делит значение переменной var на exp и присваивает результат переменной var < expl < exp2 Если exp1 меньше, чем exp2, то возвращает 1, иначе возвращает 0 << exp1 << exp2 Сдвигает exp1 влево на exp2 бит <<= var <<= exp Побитовый сдвиг влево значения переменной var на exp <= expl <= exp2 Если exp1 меньше или равно exp2, то возвращает 1; иначе возвращает 0 = var = exp Присваивает значение exp переменной var == exp1==exp2 Если exp1 равно exp2, то возвращает 1; иначе возвращает 0 > exp1 > exp2 1, если exp1 больше, чем exp2; иначе 0 >= exp1 >= exp2 1, если exp1 больше или равно exp2; иначе 0 >> exp >> exp2 Сдвигает exp1 вправо на exp2 бит >>= var >>=exp Побитовый сдвиг вправо значения переменной var на exp ^ exp1 ^ exp2 Исключающее OR выражений exp1 и exp2 ^= var ^= exp Присваивает переменной var побитовое XOR var и exp | exp1 | exp2 Побитовое OR выражений exp1 и exp2 |= var |= exp Присваивает переменной var результат операции XOR var и exp || exp1 || exp2 1, если или exp1 или exp2 являются ненулевыми значениями; иначе 0 ~ ~exp Побитовое дополнение до exp

6. Что означает операция (( ))?

Для облегчения программирования можно записывать условия оболочки bash в двойные скобки — (( )).

7. Какие стандартные имена переменных Вам известны?

– HOME — имя домашнего каталога пользователя. Если команда cd вводится без аргументов, то происходит переход в каталог, указанный в этой переменной. – IFS — последовательность символов, являющихся разделителями в командной строке, например, пробел, табуляция и перевод строки (new line). – MAIL — командный процессор каждый раз перед выводом на экран промптера проверяет содержимое файла, имя которого указано в этой переменной, и если содержимое этого файла изменилось с момента последнего ввода из него, то перед тем как вывести на терминал промптер, командный процессор выводит на терминал сообщение You have mail (у Вас есть почта). – TERM — тип используемого терминала. – LOGNAME — содержит регистрационное имя пользователя, которое устанавливается автоматически при входе в систему. В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set.

8. Что такое метасимволы?

Такие символы, как ' < > * ? | \ " &, являются метасимволами и имеют для командного процессора специальный смысл.

9. Как экранировать метасимволы?

Снятие специального смысла с метасимвола называется экранированием метасимвола. Экранирование может быть осуществлено с помощью предшествующего метасимволу символа \, который, в свою очередь, является метасимволом. Для экранирования группы метасимволов нужно заключить её в одинарные кавычки. Строка, заключённая в двойные кавычки, экранирует все метасимволы, кроме $, ' , \, ".

10. Как создавать и запускать командные файлы?

Последовательность команд может быть помещена в текстовый файл. Такой файл называется командным. Далее этот файл можно выполнить по команде: 1 bash командный_файл [аргументы]

11. Как определяются функции в языке программирования bash?

Группу команд можно объединить в функцию. Для этого существует ключевое слово function, после которого следует имя функции и список команд, заключённых в фигурные скобки.

12. Каким образом можно выяснить, является файл каталогом или обычным файлом?

Значением переменной PATH (т.е. $PATH) является список каталогов, в которых командный процессор осуществляет поиск программы или команды, указанной в командной строке, в том случае, если указанное имя программы или команды не содержит ни одного символа /.

13. Каково назначение команд set, typeset и unset?

В командном процессоре Си имеется ещё несколько стандартных переменных. Значение всех переменных можно просмотреть с помощью команды set Если использовать typeset -i для объявления и присвоения переменной, то при последующем её применении она станет целой. Удалить функцию можно с помощью команды unset с флагом -f.

14. Как передаются параметры в командные файлы?

Символ $ является метасимволом командного процессора. Он используется, в частности, для ссылки на параметры, точнее, для получения их значений в командном файле.

15. Назовите специальные переменные языка bash и их назначение.

– $* — отображается вся командная строка или параметры оболочки; – $? — код завершения последней выполненной команды; – $$ — уникальный идентификатор процесса, в рамках которого выполняется командный процессор; – $! — номер процесса, в рамках которого выполняется последняя вызванная на выполнение в командном режиме команда; – $- — значение флагов командного процессора; – ${*#} — возвращает целое число — количество слов, которые были результатом $*; – ${#name} — возвращает целое значение длины строки в переменной name; – ${name[n]} — обращение к n-му элементу массива; – ${name[*]} — перечисляет все элементы массива, разделённые пробелом; – ${name[@]} — то же самое, но позволяет учитывать символы пробелы в самих переменных; – ${name:-value} — если значение переменной name не определено, то оно будет заменено на указанное value; – ${name:value} — проверяется факт существования переменной; – ${name=value} — если name не определено, то ему присваивается значение value; – ${name?value} — останавливает выполнение, если имя переменной не определено, и выводит value как сообщение об ошибке; – ${name+value} — это выражение работает противоположно ${name-value}. Если переменная определена, то подставляется value; – ${name#pattern} — представляет значение переменной name с удалённым самым коротким левым образцом (pattern); – ${#name[*]} и ${#name[@]} — эти выражения возвращают количество элементов в массиве name.