

Лабораторная работа №12

Программирование в командном процессоре ОС UNIX. Расширенное программирование

Старовойтов Егор Сергеевич

Содержание

Цель работы	1
Задание	2
Теоретическое введение	2
Ход работы.....	3
Задание 2.....	3
Задание 3.....	4
Вывод.....	5
Контрольные вопросы	6
1. Найдите синтаксическую ошибку в следующей строке: <code>while [\$1 != "exit"]</code>	6
2. Как объединить (конкатенация) несколько строк в одну?	6
3. Найдите информацию об утилите <code>seq</code> . Какими иными способами можно реализовать её функционал при программировании на <code>bash</code> ?	6
4. Какой результат даст вычисление выражения <code>\$((10/3))</code> ?	6
5. Укажите кратко основные отличия командной оболочки <code>zsh</code> от <code>bash</code>	6
6. Проверьте, верен ли синтаксис данной конструкции <code>for ((a=1; a <= LIMIT; a++))</code>	6
7. Сравните язык <code>bash</code> с какими-либо языками программирования. Какие преимущества у <code>bash</code> по сравнению с ними? Какие недостатки?	6

Цель работы

Изучить основы программирования в оболочке ОС UNIX. Научиться писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Задание

1. Написать командный файл, реализующий упрощённый механизм семафоров. Командный файл должен в течение некоторого времени t_1 дожидаться освобождения ресурса, выдавая об этом сообщение, а дождавшись его освобождения, использовать его в течение некоторого времени $t_2 < t_1$, также выдавая информацию о том, что ресурс используется соответствующим командным файлом (процессом). Запустить командный файл в одном виртуальном терминале в фоновом режиме, перенаправив его вывод в другой (`> /dev/tty#`, где `#` — номер терминала куда перенаправляется вывод), в котором также запущен этот файл, но не фоновом, а в привилегированном режиме. Доработать программу так, чтобы имелась возможность взаимодействия трёх и более процессов.
2. Реализовать команду `man` с помощью командного файла. Изучите содержимое каталога `/usr/share/man/man1`. В нем находятся архивы текстовых файлов, содержащих справку по большинству установленных в системе программ и команд. Каждый архив можно открыть командой `less` сразу же просмотрев содержимое справки. Командный файл должен получать в виде аргумента командной строки название команды и в виде результата выдавать справку об этой команде или сообщение об отсутствии справки, если соответствующего файла нет в каталоге `man1`.
3. Используя встроенную переменную `$RANDOM`, напишите командный файл, генерирующий случайную последовательность букв латинского алфавита. Учтите, что `$RANDOM` выдаёт псевдослучайные числа в диапазоне от 0 до 32767.

Теоретическое введение

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

разработаны комитетом IEEE (Institute of Electrical and Electronics Engineers) для обеспечения совместимости различных UNIX/Linux-подобных операционных систем и переносимости прикладных программ на уровне исходного кода. POSIX-совместимые оболочки разработаны на базе оболочки Корна. Рассмотрим основные элементы программирования в оболочке bash. В других оболочках большинство команд будет совпадать с описанными ниже.

Ход работы

Задание 2

Я изучил содержимое каталога /usr/share/man/man1

```
[liveuser@localhost-live ~]$ cd /usr/share/man/man1
[liveuser@localhost-live man1]$ ls
.:1.gz
'.1.gz'
ab.1.gz
abrt.1.gz
abrt-action-analyze-backtrace.1.gz
abrt-action-analyze-c.1.gz
abrt-action-analyze-ccpp-local.1.gz
abrt-action-analyze-core.1.gz
abrt-action-analyze-java.1.gz
abrt-action-analyze-oops.1.gz
abrt-action-analyze-python.1.gz
abrt-action-analyze-vmcore.1.gz
abrt-action-analyze-vulnerability.1.gz
abrt-action-analyze-xorg.1.gz
abrt-action-check-oops-for-hw-error.1.gz
abrt-action-find-bodhi-update.1.gz
abrt-action-generate-backtrace.1.gz
abrt-action-generate-core-backtrace.1.gz
abrt-action-install-debuginfo.1.gz
abrt-action-list-dsos.1.gz
abrt-action-notify.1.gz
abrt-action-perform-ccpp-analysis.1.gz
abrt-action-save-package-data.1.gz
abrt-action-trim-files.1.gz
abrt-applet.1.gz
abrt-auto-reporting.1.gz
abrt-bodhi.1.gz
abrt-cli.1.gz
abrt-dump-journal-core.1.gz
abrt-dump-journal-oops.1.gz
abrt-dump-journal-xorg.1.gz
abrt-dump-oops.1.gz
abrt-dump-xorg.1.gz
abrt-handle-upload.1.gz
abrt-harvest-pstoreoops.1.gz
abrt-harvest-vmcore.1.gz
abrt-merge-pstoreoops.1.gz
abrt-retrace-client.1.gz
abrt-server.1.gz
abrt-watch-log.1.gz
ac.1.gz
aconnect.1.gz
addr2line.1.gz
msgfilter.1.gz
msgfmt.1.gz
msggrep.1.gz
msginit.1.gz
msgmerge.1.gz
msgunfmt.1.gz
msguniq.1.gz
mshortname.1.gz
mshowfat.1.gz
mtools.1.gz
mtoolstest.1.gz
mtrace.1.gz
mtype.1.gz
mutter.1.gz
mv.1.gz
mvxattr.1.gz
mzip.1.gz
namei.1.gz
nano.1.gz
nautilus.1.gz
nautilus-autorun-software.1.gz
nc.1.gz
ncat.1.gz
ndctl.1.gz
ndctl-activate-firmware.1.gz
ndctl-check-labels.1.gz
ndctl-check-namespace.1.gz
ndctl-clear-errors.1.gz
ndctl-create-namespace.1.gz
ndctl-destroy-namespace.1.gz
ndctl-disable-dimm.1.gz
ndctl-disable-namespace.1.gz
ndctl-disable-region.1.gz
ndctl-enable-dimm.1.gz
ndctl-enable-namespace.1.gz
ndctl-enable-region.1.gz
ndctl-freeze-security.1.gz
ndctl-init-labels.1.gz
ndctl-inject-error.1.gz
ndctl-inject-smart.1.gz
ndctl-list.1.gz
ndctl-load-keys.1.gz
ndctl-monitor.1.gz
```

/usr/share/man/man1

Я написал скрипт в файле prog2.sh, выполняющий второе задание, используя редактор vi.

```
#!/bin/bash

if test -f "/usr/share/man/man1/$1.1.gz"
then less /usr/share/man/man1/$1.1.gz
else echo "This command not found"
fi
```

Код комадного файла

Работа программы:

```
[liveuser@localhost-live ~]$ ./prog2.sh dsjdasiojdka
This command not found
```



liveuser@localhost-live:~ — /bin/bash ./prog2.sh ls

LS(1) User Commands LS(1)

ESC[1mNAMEESC[0m

ls - list directory contents

ESC[1mSYNOPSISESC[0m

ESC[1m]ls ESC[22mESC[4mOPTIONESC[24m... ESC[4mFILEESC[24m...

ESC[1mDESCRIPTIONESC[0m

List information about the FILEs (the current directory by default). Sort entries alphabetically if none of ESC[1m-cftuvSUX ESC[22mnor ESC[1m--sort ESC[22mis specified.

Mandatory arguments to long options are mandatory for short options too.

ESC[1m-aESC[22m, ESC[1m--allESC[0m

do not ignore entries starting with .

ESC[1m-AESC[22m, ESC[1m--almost-allESC[0m

do not list implied . and ..

ESC[1m--authorESC[0m

with ESC[1m-lESC[22m, print the author of each file

ESC[1m-bESC[22m, ESC[1m--escapeESC[0m

print C-style escapes for nongraphic characters

ESC[1m--block-sizeESC[22m=ESC[4mSIZEESC[0m

with ESC[1m-lESC[22m, scale sizes by SIZE when printing them; e.g., '--block-size=M'; see SIZE format below

ESC[1m-BESC[22m, ESC[1m--ignore-backupsESC[0m

do not list implied entries ending with ~

ESC[1m-c ESC[22mwith ESC[1m-lESC[22m: sort by, and show, ctime (time of last modification of file status information); with ESC[1m-lESC[22m: show ctime and sort by name; otherwise: sort by ctime, newest first

ESC[1m-C ESC[22mlist entries by columns

ESC[1m--colorESC[22m[=ESC[4mWHENESC[24m]

colorize the output; WHEN can be 'always' (default if omitted), 'auto', or 'never'; more info below

ESC[1m-dESC[22m, ESC[1m--directoryESC[0m

list directories themselves, not their contents

/usr/share/man/man1/ls.1.gz

Задание 3

Я написал скрипт в файле prog3.sh, выполняющий третье задание, используя редактор vi.

```
#!/bin/bash
echo $RANDOM | tr '0-9' 'A-Z'
[liveuser@localhost-live ~]$
```

Код командного файла

```
[liveuser@localhost-live ~]$ touch prog3.sh
[liveuser@localhost-live ~]$ chmod +x prog3.sh
[liveuser@localhost-live ~]$ vi prog3.sh
[liveuser@localhost-live ~]$ ./prog3.sh
IDEF
[liveuser@localhost-live ~]$ ./prog3.sh
BCHAC
[liveuser@localhost-live ~]$ ./prog3.sh
BEFFD
[liveuser@localhost-live ~]$ ./prog3.sh
CHHCA
[liveuser@localhost-live ~]$ ./prog3.sh
DAHGH
[liveuser@localhost-live ~]$ ./prog3.sh
BFAHF
[liveuser@localhost-live ~]$ ./prog3.sh
CCDAH
[liveuser@localhost-live ~]$ ./prog3.sh
DAEEI
[liveuser@localhost-live ~]$ ./prog3.sh
CBCCG
[liveuser@localhost-live ~]$ ./prog3.sh
JAGC
[liveuser@localhost-live ~]$ ./prog3.sh
CBIBH
[liveuser@localhost-live ~]$
```

Работа программы

Вывод

Я изучил основы программирования в оболочке ОС UNIX. Научился писать более сложные командные файлы с использованием логических управляющих конструкций и циклов.

Контрольные вопросы

1. Найдите синтаксическую ошибку в следующей строке: `while [$1 != "exit"]`

Не хватает пробелов после `[` и до `]`.

2. Как объединить (конкатенация) несколько строк в одну?

С помощью оператора `+` над строковым типом, например

```
s1 = "Te"  
s1 += "xt"
```

После чего `s1` будет содержать "Text".

3. Найдите информацию об утилите `seq`. Какими иными способами можно реализовать её функционал при программировании на `bash`?

Команда `seq` выводит последовательность целых или действительных чисел, подходящую для передачи в другие программы. Команда `seq` может пригодиться в различных других командах и циклах для генерации последовательности чисел.

Команду `seq` можно реализовать с помощью цикла `while`.

4. Какой результат даст вычисление выражения `$((10/3))`?

3, так как это целочисленное деление без остатка.

5. Укажите кратко основные отличия командной оболочки `zsh` от `bash`.

В `zsh` есть хеш-таблицы, числа с плавающей точкой, разделение экрана, встроенный калькулятор `zcalc`.

6. Проверьте, верен ли синтаксис данной конструкции `for ((a=1; a <= LIMIT; a++))`

Синтаксис верен.

7. Сравните язык `bash` с какими-либо языками программирования. Какие преимущества у `bash` по сравнению с ними? Какие недостатки?

Плюсы: - `Bash` распространен и универсален - С его помощью можно автоматизировать работу в терминале - Удобное управление вводом/выводом - Возможность легкого вызова сторонних программ

Минусы: - Непереносимость кода на другие операционные системы - Другие языки могут расширять свои возможности с помощью библиотек, а `bash` только с помощью установки программного обеспечения.