Санкт-Петербургский Политехнический Университет Петра Великого

Кафедра компьютерных систем и программных технологий

Отчёт по лабораторной работе №1

Курс: «Операционные системы»

Тема: «Интерпретаторы командной строки Linux»

Выполнил студент:

Бояркин Никита Сергеевич
Группа: 43501/3

Проверил:

Душутина Елена Владимировна

Санкт-Петербург
2016 г.

Содержание

Лабораторная работа №1

1.1 Цель работы

- Изучение основных команд пользовательского интерфейса.
- Изучение цикла подготовки и исполнения программ.
- Изучение команд и утилит обработки текстов.

1.2 Программа работы

- Команды работы с файлами.
- Цикл работы программы (компиляторы, запуск отладчика).
- Файловый состав ОС.
- Интерпретаторы (как работают, где есть).
- Запуск script.
- Утилиты и фильтры.
- Организация конвейера.

1.3 Ход работы

1.3.1 Команды работы с файлами

Команда ls

Изучим команду получения информации о файлах и папках *ls*:

```
nikita@nikita-pc:~/Desktop$ ls
clion.sh
files
firefox.desktop
gedit.desktop
gnome-mines.desktop
gnome-sudoku.desktop
gnome-system-monitor-kde.desktop
libreoffice-writer.desktop
Matlab.sh
projects
Quartus II.desktop
software
student
texstudio.desktop

nikita@nikita-pc:~/Desktop$ ls files
projects  software  student

nikita@nikita-pc:~/Desktop$ ls -l files
lrwxrwxrwx 1 nikita nikita 18 июн 11 20:39 files -> /home/nikita/files

nikita@nikita-pc:~/Desktop$ ls Quartus\ II.desktop
Quartus II.desktop
```

Команда *ls* без ключей и указания абсолютного пути к директории выведет все файлы и папки в текущем каталоге. Если используется ключ -l, то выводится информация в виде таблицы: в первом столбце выводятся права доступа для пользователя, группы и остальных на файлы и каталоги (r - чтение, w - запись, x - выполнение). В следующих столбцах выводится количество ссылок на файл или каталог, имя владельца и имя группы, размер в байтах, дата последней модификации и самого файла или каталога имя. Если задать параметром не директорию, а файл, то тогда команда выведет только название этого файла.

Команда cat

Изучим команду просмотра содержимого файлов *cat*:

```
nikita@nikita-pc:~/Desktop$ cat Quartus\ II.desktop [Desktop Entry]
Type=Application
Version=0.9.4
Name=Quartus II 13.1 (64-bit) Web Edition
Comment=Quartus II 13.1 (64-bit)
Icon=/home/nikita/files/software/altera/13.1/quartus/adm/quartusii.png
Exec=/home/nikita/files/software/altera/13.1/quartus/bin/quartus --64bit
Terminal=false
Path=/home/nikita/files/software/altera/13.1
Name[en_US]=Quartus II

nikita@nikita-pc:~/Desktop$ cat files
cat: files: Is a directory

nikita@nikita-pc:~/Desktop$ cat Quartus\ II.desktop gedit.desktop

[Desktop Entry]
Type=Application
(...)
Path=/home/nikita/files/software/altera/13.1
Name[en_US]=Quartus II

```

```
#!/usr/bin/env xdg-open
[Desktop Entry]
Name=gedit
(...)
[Desktop Action new-document]
Name=Open a New Document
Exec=gedit --new-document
```

Команда *cat* выводит содержимое указанных файлов (одного или нескольких) или печатает сообщение об ошибке, если параметром указана директория.

Команды mv, cp

Исследуем команду *mv*, которая переименовывает или перемещает файл, и команду *cp*, которая копирует файл в указанную директорию:

```
nikita@nikita-pc:~$ mv temp.file temp1.txt
nikita@nikita-pc:~$ ls temp1.txt
temp1.txt
nikita@nikita-pc:~$ cat temp.file
cat: temp.file: No such file or directory
nikita@nikita-pc:~$ cp temp1.txt Desktop/
nikita@nikita-pc:~$ ls Desktop/
clion.sh
files
firefox.desktop
gedit.desktop
gnome-mines.desktop
gnome-sudoku.desktop
gnome-system-monitor-kde.desktop
libreoffice-writer.desktop
Matlab.sh
projects
Quartus II.desktop
software
student
temp1.txt
( ... )
```

Команда *mv* действительно переименовала файл, а после применения команды *cp* файл также отображается в списке директории, в которую мы скопировали файл.

Команды pwd, cd, mkdir, rmdir, rm

Исследуем процесс навигации по файловой системе, исследования системных папок и создание/удаление новых файлов и папок:

```
nikita@nikita-pc:~$ pwd
/home/nikita
nikita@nikita-pc:~$ >prog.c
nikita@nikita-pc:~$ ls /home/nikita
Desktop     examples.desktop  Pictures  temp1.txt
Documents   files             prog.c    Templates
Downloads   Music             Public    Videos
nikita@nikita-pc:~$ ls /
bin     initrd.img      media  sbin  var
boot    initrd.img.old  mnt    snap  vmlinuz
cdrom   lib             opt    srv   vmlinuz.old
dev     lib32           proc   sys
etc     lib64           root   tmp
home    lost+found      run    usr
nikita@nikita-pc:~$ ls -l /
total 104
drwxr-xr-x   2 root root  4096 май 29 23:06 bin
drwxr-xr-x   3 root root  4096 июл 30 14:27 boot
drwxrwxr-x   2 root root  4096 май 29 22:57 cdrom
drwxr-xr-x  21 root root  4320 сен 13 10:12 dev
```

```
( ... )
drwxr-xr-x  12 root root  4096 июл 30 10:37 usr
drwxr-xr-x  14 root root  4096 апр 21 01:19 var
lrwxrwxrwx   1 root root    29 июл 30 14:25 vmlinuz -> boot/vmlinuz-4.4.0-31-generic
lrwxrwxrwx   1 root root    29 май 29 23:00 vmlinuz.old -> boot/vmlinuz-4.4.0-21-generic
nikita@nikita-pc:~$ ls /bin
bash            networkctl
bunzip2         nisdomainname
busybox         ntfs-3g
bzcat           ntfs-3g.probe
bzcmp           ntfs-3g.secaudit
bzdiff          ntfs-3g.usermap
( ... )

nikita@nikita-pc:~$ cd ..
nikita@nikita-pc:/home$ cd ..
nikita@nikita-pc:/$ cd ..
nikita@nikita-pc:/$ ls -ld /
drwxr-xr-x 25 root root 4096 июл 30 14:25 /
nikita@nikita-pc:/$ ls -ld /bin
drwxr-xr-x 2 root root 4096 май 29 23:06 /bin
nikita@nikita-pc:/$ ls -ld /home/nikita/
drwxr-xr-x 25 nikita  nikita 4096 сен 13 10:53 /home/nikita/
nikita@nikita-pc:/$ cd
nikita@nikita-pc:~$ mkdir temp
nikita@nikita-pc:~$ cd temp
nikita@nikita-pc:~/temp$ cp ../prog.c .
nikita@nikita-pc:~/temp$ cd ..
nikita@nikita-pc:~$ ls
Desktop            files      Public     Videos
Documents          Music      temp
Downloads          Pictures   temp1.txt
examples.desktop   prog.c     Templates
nikita@nikita-pc:~$ rmdir temp/
rmdir: failed to remove 'temp/': Directory not empty
nikita@nikita-pc:~$ rm temp/prog.c
nikita@nikita-pc:~$ rmdir temp/
nikita@nikita-pc:~$ ls
Desktop    examples.desktop  Pictures  temp1.txt
Documents  files             prog.c    Templates
Downloads  Music             Public    Videos
```

- *pwd* - выводит полное имя текущей директории.
- *>filename* - создает новый файл в текущей директории.
- *ls /* - выводит имена файлов и папок в корне файловой системы.
- *cd* - меняет текущую директорию (можно указывать как полный путь к директории, так и относительный). Команда использует ".."как указатель на директорию выше по иерархии и ". как указатель на текущую директорию (что работает и для остальных команд). При попытке подняться выше, чем корень файловой системы, команда ничего не сделает.
- *mkdir* - создает новую директорию.
- *rmdir* - удаляет пустую директорию. Если директория не пуста, то сначала н используют команду *rm dir/**, которая удалит все файлы внутри.
- *rm* - удаляет файл или множество файлов, с помощью символа "*".

Команда ls с флагом -l

ls -l - выводит имена файлов и папок с дополнительной информацией о типе файла, правах доступа к файлу, количестве ссылок на файл, имени владельца, имени группы, размере файла (в байтах) и временном штампе.

Был проведен эксперимент для изучения вывода команды *ls -l*:

```
nikita@nikita-pc:~/temp$ mkdir example
nikita@nikita-pc:~/temp$ cd example
nikita@nikita-pc:~/temp/example$ >cat.ty
nikita@nikita-pc:~/temp/example$ ls -l
total 0
-rw-rw-r-- 1 nikita nikita 0 сен 23 12:28 cat.ty
nikita@nikita-pc:~/temp/example$ ln cat.ty ../cat.ty
nikita@nikita-pc:~/temp/example$ ls -l
total 0
-rw-rw-r-- 2 nikita nikita 0 сен 23 12:28 cat.ty
nikita@nikita-pc:~/temp/example$ rm ../cat.ty
nikita@nikita-pc:~/temp/example$ ls -l
total 0
-rw-rw-r-- 1 nikita nikita 0 сен 23 12:28 cat.ty
nikita@nikita-pc:~/temp/example$ mkdir folder
nikita@nikita-pc:~/temp/example$ ls -l
total 4
-rw-rw-r-- 1 nikita nikita    0 сен 23 12:28 cat.ty
drwxrwxr-x 2 nikita nikita 4096 окт  6 02:57 folder
nikita@nikita-pc:~/temp/example$ ls -as
total 12
4 .  4 ..  0 cat.ty  4 folder
nikita@nikita-pc:~/temp/example$ echo "somesomesomesomesomesomesome">cat.ty
nikita@nikita-pc:~/temp/example$ ls -as
total 16
4 .  4 ..  4 cat.ty  4 folder
```

В начале эксперимента создана пустая папка *example* и пустой файл *cat.ty* внутри. Далее исследуем содержимое командой *ls -l*, в результате чего получено: *total 0 -rw-rw-r– 1 nikita nikita 0 сен 23 12:28 cat.ty*. Разберем вывод по частям:

- *total 0* - общее дисковое пространство, выраженное в блоках, используемое всеми файлами в данном каталоге. В ходе эксперимента было выявлено увеличение и уменьшение количество занимаемых блоков, в зависимости от размера содержимого файлов. Более наглядную информацию об этом параметре можно получить командой *ls -as*.
- *-rw-rw-r–* - права доступа для владельца (rw- чтение и запись), для группы (rw- чтение и запись) и для остальных (r– чтение). Параметр можно изменить с помощью команды *chmod*.
- *1* - количество жестких ссылок на файл. Экспериментально подтверждено, что параметр можно изменить, создав дополнительную ссылку, с помощью команды *ln*.
- *nikita nikita* - имя владельца и группы.
- *0* - размер файла в байтах. Было установлено, что для только что созданного каталога, по умолчанию, размер будет равен 4096 байт (4 блока).
- *сен 23 12:28* - временной штамп.
- *cat.ty* - символьное имя файла.

Последующая часть эксперимента была направлена на изучение этих полей.

Команда ps

Исследуем команду *ps*, которая выводит информацию о работающих процессах:

```
nikita@nikita-pc:~$ ps
  PID TTY          TIME CMD
 3274 pts/4    00:00:00 bash
 3294 pts/4    00:00:00 ps
nikita@nikita-pc:~$ ps -A
  PID TTY          TIME CMD
    1 ?        00:00:04 systemd
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    7 ?        00:00:03 rcu_sched
    8 ?        00:00:00 rcu_bh
    9 ?        00:00:00 migration/0
   10 ?        00:00:00 watchdog/0
   11 ?        00:00:00 watchdog/1
   12 ?        00:00:00 migration/1
   ( ... )
 3267 ?        00:00:01 gnome-terminal-
 3274 pts/4    00:00:00 bash
 3295 pts/4    00:00:00 ps
nikita@nikita-pc:~$ ps -T
  PID  SPID TTY          TIME CMD
 2278  2278 pts/2    00:00:00 bash
 2295  2295 pts/2    00:00:00 ps
nikita@nikita-pc:~$ ps r
  PID TTY      STAT   TIME COMMAND
 2347 pts/2    R+     0:00 ps r
nikita@nikita-pc:~$ ps -d
  PID TTY          TIME CMD
    2 ?        00:00:00 kthreadd
    3 ?        00:00:00 ksoftirqd/0
    5 ?        00:00:00 kworker/0:0H
    6 ?        00:00:01 kworker/u8:0
    7 ?        00:00:01 rcu_sched
    ( ... )
 2104 ?        00:00:00 dhclient
 2198 ?        00:00:00 avahi-daemon
 2271 ?        00:00:04 gnome-terminal-
 2325 ?        00:00:00 gvfsd-http
 2335 ?        00:00:03 gedit
 2354 pts/2    00:00:00 ps
nikita@nikita-pc:~$ ps -N -d
  PID TTY          TIME CMD
    1 ?        00:00:04 systemd
  232 ?        00:00:00 systemd-journal
  265 ?        00:00:01 systemd-udevd
    ( ... )
 1518 ?        00:00:00 udisksd
 1522 ?        00:00:00 colord
 2178 ?        00:00:00 avahi-daemon
 2278 pts/2    00:00:00 bash
```

Выводимые столбцы:

- *PID* - уникальный идентификатор процесса.
- *TTY* - терминал, с которым связан данный процесс.
- *TIME* - процессорное время, занятое этим процессом.
- *CMD* - команда, запустившая данный процесс «с некоторыми опциями выводит и каталог откуда процесс был запущен».

Рассмотренные ключи команды:

- *-A* - выводит информацию обо всех процессах.
- *-T* - выводит информацию обо всех процессах на конкретном терминале.
- *r* - выводит информацию исключительно о работающих процессах.
- *-d* - выводит информацию обо всех процессах, кроме главных системных процессов сеанса.
- *-N* - отрицание выбора.

1.3.2 Цикл работы программы (компиляторы, запуск отладчика)

Исследуем процесс компиляции программ на языке "C":

```
nikita@nikita-pc:~$ cd temp
nikita@nikita-pc:~/temp$ >prog.c
nikita@nikita-pc:~/temp$ gedit prog.c
nikita@nikita-pc:~/temp$ gcc prog.c -c
prog.c:2:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ()
 ^
nikita@nikita-pc:~/temp$ gcc prog.o -o prog
nikita@nikita-pc:~/temp$ ./prog
Hello, everybody !
nikita@nikita-pc:~/temp$ ./prog >res.txt
nikita@nikita-pc:~/temp$ cat res.txt
Hello, everybody !

nikita@nikita-pc:~/temp$ >my_open.c
nikita@nikita-pc:~/temp$ gedit my_open.c
nikita@nikita-pc:~/temp$ gcc my_open.c -o my_open
my_open.c:2:1: warning: return type defaults to ‘int’ [-Wimplicit-int]
 main ( argc, argv )
 ^
nikita@nikita-pc:~/temp$ ./my_openоткрываемый
 файл не указан
nikita@nikita-pc:~/temp$ ./my_open res.txt
./my_open: файл res.txt открыт
nikita@nikita-pc:~/temp$ ./my_open blahblah
./my_open: неудача при попытке открыть файл blahblah
```

prog.c:

```
#include <stdio.h>
main ()
{
  printf ("Hello, everybody ! ");
}
```

my_open.c:

```
#include <stdio.h>
main ( argc, argv )
  int argc;
  char *argv [];
{
  if (argc > 1) {
     if (fopen(argv[1],"r") == NULL)
    printf("%s: неудача при попытке открыть файл %s \n",argv[0],argv[1]);
    else
    printf("%s: файл %s открыт \n",argv[0],argv[1]);
  }
  else
       printf("%s: открываемый файл не указан \n", argv[0]);
}
```

В первую очередь были созданы 2 файла с текстами программ ("*>prog.c*, *>my_ open.c*"), после этого были созданы объектные файлы, с помощью компилятора *gcc* и флага *-c*, который подавляет действие компоновщика связей (*ld*). На заключительном этапе, с помощью компилятора *gcc* и флага *-o*, производится вызов компоновщика связей и создается исполняемый файл. В дальнейшем, принудительное создание объектного файла, с помощью флага *-c* игнорируется и сразу компилируется исполняемый файл.

Запуск программы на исполнение производится с помощью конструкции *./progname*, где "."означает текущий каталог. Перенаправление выходного потока программы в файл *res.txt* осуществляется с помощью команды *./progname >res.txt*.

С помощью аргументов функции main(int argc, char *argv []) можно обрабатывать информацию из консоли (вызывающего окружения). С помощью команды *./my_ open res.txt* была передана строка "res.txt"программе my_open. Если ничего не было передано или неверное имя файла для обработки, то выводится ошибка.

Попробуем скомпоновать программу из нескольких файлов, для этого разработаем тестовую программу. prog.c:

```
#include <stdio.h>

extern int foo();

int main()
{
  printf ("Hello, everybody ! Number from extern function: %d", foo());
  return 0;
}
```

foo.c:

```
int foo() {
  return 202;
}
```

Функция *foo()* объявлена как внешняя, поэтому можно осуществить компоновку из внешнего файла:

```
nikita@nikita-pc:~/temp$ gcc -c prog.c foo.c
nikita@nikita-pc:~/temp$ gcc prog.o foo.o -o result
nikita@nikita-pc:~/temp$ ./result
Hello, everybody ! Number from extern function: 202
```

Для выявления ошибок при компиляции можно исследовать вывод компилятора *gcc*, где указывается подробная информация об ошибках и предупреждениях. Для отладки программы можно использовать отладчик *gdb*. Отладчик *gdb* – интерактивный, поэтому его можно запускать без параметров, а задавать их прямо из него. Если указано имя исполнимого файла, то он будет сразу же загружен в отладчик.

Создадим тестовую программу и попробуем ее отладить: gdbtest.c:

```
#include <stdio.h>

int main() {
  int num;
  printf("hello\n");
  scanf("%d", &num);
  printf("num %d\n", num);
  return 0;
}
```

Запустим отладчик gdb:

```
nikita@nikita-pc:~/temp$ gdb gdbtest
GNU gdb (Ubuntu 7.11-0ubuntu1) 7.11
( ... )
Reading symbols from gdbtest...(no debugging symbols found)...done.
(gdb) break main
Breakpoint 1 at 0x40064a
(gdb) run -v
Starting program: /home/nikita/temp/gdbtest -v

Breakpoint 1, 0x000000000040064a in main ()
(gdb) next
Single stepping until exit from function main,
```

```
13 which has no line number information.
14 hello
15 5
16 num 5
17 __libc_start_main (main=0x400646 <main>, argc=2, argv=0x7fffffffde68,
18     init=<optimized out>, fini=<optimized out>, rtld_fini=<optimized out>,
19     stack_end=0x7fffffffde58) at ../csu/libc-start.c:325
20 325 ../csu/libc-start.c: No such file or directory.
21 (gdb) quit
```

При отладке программы была поставлена точка останова на функции *main()*. После этого был произведен запуск программы и отладчик уведомил о том, что программа постигла точки останова по адресу функции *main()*. Продолжение программы осуществляется командой *next*. После этого предлагается ввести число, это можно сделать прямо из отладчика и программа продолжится. Результат работы программы тоже выводится в отладчике. Для выхода из отладчика используется команда *quit*.

1.3.3 Файловый состав ОС

Рассмотрим содержимое основных системных каталогов:

- */bin* - этот каталог содержит в основном готовые к исполнению программы, большинство из которых необходимы во время старта системы (или в однопользовательском системном режиме, используемом для отладки). Здесь хранится значительное количество общеупотребительных команд Linux.
- */boot* - содержит основные постоянные файлы для загрузки системы, в частности загружаемое ядро. Файлы из этого каталога нужны только во время загрузки системы.
- */dev* - каталог специальных файлов или файлов устройств.
- */etc* - этот каталог и его подкаталоги содержат большинство данных, необходимых для начальной загрузки системы и основные конфигурационные файлы. Каталог /etc не должен содержать двоичных файлов (их следует перенести в /bin или /sbin).
- */home* - в этом каталоге находятся домашние каталоги пользователей.
- */lib* - этот каталог содержит разделяемые библиотеки функций, необходимых компилятору языка C и модули (драйверы устройств). Даже если в системе не установлен компилятор языка C, разделяемые библиотеки необходимы, поскольку они используются многими прикладными программами. Они загружаются в память по мере необходимости выполнения каких-то функций, что позволяет уменьшить объем кода программ — в противном случае один и тот же код многократно повторялся бы в различных программах.
- */root* - в этом каталоге находится домашний каталог суперпользователя.
- */tmp* - каталог для временных файлов. В любой момент суперпользователь может удалить файлы из этого каталога без большого ущерба для остальных пользователей.
- */usr* - в его подкаталогах находятся все основные приложения. В соответствии со стандартом FHS рекомендуется выделять для этого каталога отдельный раздел диска или вообще располагать его на сетевом диске, общем для всех компьютеров в сети. Такой раздел или диск монтируют только для чтения и располагают в нем общие конфигурационные и исполняемые файлы, документацию, системные утилиты и библиотеки, а также включаемые файлы.
- */usr/bin* - готовые к исполнению программы — утилиты и приложения, которые часто вызывают обычные пользователи.
- */usr/etc* - здесь содержатся конфигурационные файлы для группы машин. Однако, команды и программы должны смотреть в каталог /etc, в котором должны быть линки к файлам в каталоге /usr/etc.
- */usr/etc/include* - этот каталог содержит исходный код стандартных библиотек языка C, подставляемый в программы директивой препроцессора include.
- */usr/lib* - в данном каталоге содержится объектные библиотеки подпрограмм, динамические библиотеки, некоторые готовые к исполнению программы, которые не вызываются непосредственно. Сложные программные системы могут иметь свои подкаталоги в этом каталоге.

- */usr/local* - здесь помещают программы и подкаталоги, которые являются локальными (уникальными) для данной машины. Также содержит подкаталоги */bin*, */etc*, */lib*, */man*, */src*, которые отличаются от уже рассмотренных тем, что они являются уникальными для данной машины.
- */usr/man* - страницы интерактивного руководства man в исходном формате (не подготовленные для просмотра).
- */usr/src* - Исходные тексты для различных частей Linux. */usr/src/linux* — исходные тексты для ядра Linux.
- */usr/tmp* - еще одно место для хранения временных файлов.
- */var* - Этот каталог содержит файлы, в которых сохраняются различные переменные данные, определяющие конфигурацию некоторых программ при следующем запуске или временно сохраняемую информацию, которая будет использоваться позже в ходе текущего сеанса. Объем данных в этом каталоге может сильно изменяться, поскольку он содержит, например, файлы протоколов (логи), файлы спулинга и блокировки (locking), временные файлы и т.д.

1.3.4 Интерпретаторы

Интерпретатор командной строки - компьютерная программа, часть операционной системы, обеспечивающая базовые возможности управления компьютером посредством интерактивного ввода команд через интерфейс командной строки или последовательного исполнения пакетных командных файлов.

Зачастую интерпретатор командной строки предоставляет возможность использования циклов, операторов условного и безусловного перехода и переменных. Он позволяет писать как несложные сценарии для автоматизации повседневных задач, так и довольно сложные программы.

Примеры интерпретаторов: для DOS - *command.com*, для Windows - *cmd.exe*, *PowerShell*, для UNIX - *bash*, *csh*, *ksh*, *zsh*.

Узнаем какой именно интерпретатор используем с помощью команды *ps -T* и запустим другой интерпретатор непосредственно из первого:

```
nikita@nikita-pc:~$ ps -T
  PID  SPID TTY          TIME CMD
 3464  3464 pts/2    00:00:00 bash
 3475  3475 pts/2    00:00:00 ps
nikita@nikita-pc:~$ sudo apt install csh
( ... )
nikita@nikita-pc:~$ csh
% who
nikita   tty7         2016-10-04 22:47 (:0)
% ps -T
  PID  SPID TTY          TIME CMD
 2738  2738 pts/3    00:00:00 bash
 3385  3385 pts/3    00:00:00 csh
 3432  3432 pts/3    00:00:00 ps
%
```

Можно заметить, что команда *ps -T* выдаст информацию не только о текущем интерпретаторе, но и обо всех процессах запущенных им.

1.3.5 Запуск script

Для целей лабораторных работ был разработан скрипт, который сохраняет информацию из командной строки в файл:

script.sh:

```
#!/bin/bash

filename=$1
dir=$HOME/$filename.txt
command=s

# prompt="$(whoami)@$(hostname): $(/bin/pwd | sed "s|$HOME|~|") $"

> $dir

```

```
echo "Log to $dir enabled, old content removed"
echo "Execute "q" or press ^C to log off"

while [ "$command" != q ]
do
  printf "$(whoami)@$(hostname): $(/bin/pwd | sed "s|$HOME|~|") $ "
  read command
  ifcd="$(echo "$command" | cut -f 1 -d " ")"
  if [ "$ifcd" = "cd" ];
  then
    path=$(echo "$command" | sed "s/cd //")
    echo "$(whoami)@$(hostname): $(/bin/pwd | sed "s|$HOME|~|") $ cd $path" >> $dir
    cd $path
  else
    echo "$(whoami)@$(hostname): $(/bin/pwd | sed "s|$HOME|~|") $ $command" >> $dir
    result=$($command)
    echo "$result" >> $dir
    echo "$result"
  fi
done

echo "Log closed"
```

Недостатком данного скрипта является отсутствие автоподстановки команд, которая существенно экономит время работы, поэтому этот метод не является универсальным. В тоже время простое копирование из командной строки занимает не намного больше времени, зато позволяет использовать все удобства при работе.

1.3.6 Утилиты и фильтры

Утилиты

Утилита - это сервисная программа, облегчающая пользование другими программами, работу с компьютером. Рассмотрим некоторые полезные утилиты.

Введем набор команд для получения информации об ОС и текущем сеансе:

```
nikita@nikita-pc:~$ dateВт
 сен 13 10:24:21 MSK 2016
nikita@nikita-pc:~$ who
nikita   tty7         2016-09-13 10:12 (:0)
nikita@nikita-pc:~$ whoami
nikita
nikita@nikita-pc:~$ tty
/dev/pts/18
nikita@nikita-pc:~$ logname
logname: no login name
nikita@nikita-pc:~$ uname
Linux
```

- *date* - выводит информацию о текущем системном времени.
- *who* - выводит пользователей системы, которые в данный момент находятся в ней.
- *whoami* - выводит имя пользователя, ассоциированное с текущим эффективным идентификатором пользователя.
- *tty* - выводит на экран полное имя файла-терминала.
- *logname* - выводит имя пользователя, под которым он произвел вход в систему.
- *uname* - выводит на экран имя UNIX-системы.

Изучим команду задержки на указанное время:

```
nikita@nikita-pc:~$ sleep 5
nikita@nikita-pc:~$ sleep 10000
^C
```

sleep - задерживает на указанное время (задается в секундах, однако можно задавать в часах, например 5h).

Команду *sleep* (как и другие) можно преждевременно остановить, послав сигнал прерывания, с помощью комбинации клавиш *Ctrl+C*.

С помощью команды *man*, можно получить справочную информацию о любой команде в формате справочника:

```
nikita@nikita-pc:~$ man date
```

Справочники в ОС Linux имеют следующие разделы:

- *NAME* - указывается название команды и ее функциональное применение.
- *SYNOPSIS* - указывается синтаксис команды (все что не заключено в квадратные скобки обязательно к добавлению).
- *DESCRIPTION* - описание флагов программы.
- *EXAMPLES* - примеры работы команды.
- *AUTHOR* - автор программы.
- *REPORTING BUGS* - контактные данные для обращения по поводу выявленных ошибок.
- *COPYRIGHT* - информация о лицензии.
- *SEE ALSO* - список похожих команд, рекомендованных для просмотра.

Фильтры

Существует большое число команд UNIX, которые читают входной поток, выполняют простые операции над ним и записывают результат в выходной поток. Такие программы называются фильтрами. Многие команды могут быть фильтрами, поскольку обычно в случае незадания файла-аргумента читается стандартный ввод. Важной особенностью фильтров является то, что они никогда не изменяют исходных файлов, а лишь выводят на стандартный вывод обработанную информацию. Стандартный вывод также можно переназначить в файл.

Фильтр Grep

Изучим команду *grep*, которая осуществляет поиск по шаблону, заданному регулярным выражением. Для этого выполним тестовое задание:

```
Выведите только те строки из вывода ls -l /tmp, которые:
1) соответствуют каталогам;
2) соответствуют выполняемым для всех файлам;
3) принадлежат пользователь root;
4) не принадлежат пользователю root.
```

Результат фильтрации представлен в листинге:

```
nikita@nikita-pc:~/temp$ ls -l /tmp | grep '^d'
drwx------ 2 nikita nikita 4096 сен 13 12:30 lu493719xsav.tmp
drwx------ 3 root   root   4096 сен 13 10:12 systemd-private-7825063278
    dd42f399ad5f71661fe3a7-colord.service-Kpn7Wo
drwx------ 3 root   root   4096 сен 13 10:12 systemd-private-7825063278
    dd42f399ad5f71661fe3a7-rtkit-daemon.service-apgp9Y
drwx------ 3 root   root   4096 сен 13 10:12 systemd-private-7825063278
    dd42f399ad5f71661fe3a7-systemd-timesyncd.service-oyLPji
nikita@nikita-pc:~/temp$ ls -l /tmp | grep '.........x'
drwx------ 2 nikita nikita 4096 сен 13 12:30 lu493719xsav.tmp
srwxrwxr-x 1 nikita nikita    0 сен 13 12:26
    OSL_PIPE_1000_SingleOfficeIPC_d6a98e563430a207ff34195dc355b
nikita@nikita-pc:~/temp$ ls -l /tmp | grep 'root'
drwx------ 3 root   root   4096 сен 13 10:12 systemd-private-7825063278
    dd42f399ad5f71661fe3a7-colord.service-Kpn7Wo
drwx------ 3 root   root   4096 сен 13 10:12 systemd-private-7825063278
    dd42f399ad5f71661fe3a7-rtkit-daemon.service-apgp9Y
```

```
drwx------ 3 root   root   4096 сен 13 10:12 systemd-private-7825063278
    dd42f399ad5f71661fe3a7-systemd-timesyncd.service-oyLPji
nikita@nikita-pc:~/temp$ ls -l /tmp | grep -v 'root'
total 16
-rw------- 1 nikita nikita    0 сен 13 10:12 config-err-OJoivp
drwx------ 2 nikita nikita 4096 сен 13 12:30 lu493719xsav.tmp
srwxrwxr-x 1 nikita nikita    0 сен 13 12:26
    OSL_PIPE_1000_SingleOfficeIPC_d6a98e563430a207ff34195dc355b
-rw-rw-r-- 1 nikita nikita    0 сен 13 10:12 unity_support_test.0
```

Фильтр Cut

Изучим команду *Cut*, которая выбирает отдельные поля из строк файла. Для этого выполним тестовое задание:

```
Определите с использованием команды cut:
1) номера запущенных Вами процессов;
2) идентификаторы пользователей,  имеющих  x-терминалы на Вашей рабочей станции;
3) Ваше входное имя в системе.
```

Результат фильтрации представлен в листинге:

```
nikita@nikita-pc:~/temp$ ps
  PID TTY          TIME CMD
 5365 pts/4    00:00:00 bash
 6031 pts/4    00:00:00 ps
nikita@nikita-pc:~/temp$ ps | cut -f-2 -d " "

 5365
 6035
 6036
nikita@nikita-pc:~/temp$ who
nikita   tty7         2016-09-13 10:12 (:0)
nikita@nikita-pc:~/temp$ who | cut -f1 -d ' '
nikita
nikita@nikita-pc:~/temp$ whoami
nikita
nikita@nikita-pc:~/temp$ whoami | cut -f1 -d ' '
nikita
```

Фильтр Tr

Изучим команду *Tr*, которая копирует стандартный ввод на стандартный вывод с заменой либо удалением выбранных символов. Символы, найденные в цепочке1, заменяются на соответствующие символы из цепочки2. Для этого выполним тестовое задание:

```
Выведите информацию о ваших файлах прописными буквами.
```

Результат фильтрации представлен в листинге:

```
nikita@nikita-pc:~/temp$ ls
file_name   my_open.c  prog     res.txt
my_open     my_open.o  prog.c
nikita@nikita-pc:~/temp$ ls | tr [a-z] [A-Z]
FILE_NAME
MY_OPEN
MY_OPEN.C
MY_OPEN.O
PROG
PROG.C
RES.TXT
```

Фильтр Sort

Изучим команду *Sort*, которая сортирует строки, входящие во все исходные файлы, и выдает результат на стандартный вывод. Для этого выполним тестовое задание:

```
Отсортируйте файлы в вашем каталоге (ls -l):
1) в алфавитном порядке,
2) в порядке увеличения размеров файлов,
3) в порядке уменьшения размеров файлов.
```

Результат фильтрации представлен в листинге:

```
nikita@nikita-pc:~/temp$ ls -l
total 52
-rwxrwxr-x 1 nikita nikita 8656 сен 13 13:02 file_name
-rwxrwxr-x 1 nikita nikita 8656 сен 13 13:02 my_open
-rw-rw-r-- 1 nikita nikita  397 сен 13 13:02 my_open.c
-rw-rw-r-- 1 nikita nikita 1960 сен 13 12:45 my_open.o
-rwxrwxr-x 1 nikita nikita 8608 сен 13 12:54 prog
-rw-rw-r-- 1 nikita nikita   68 сен 13 11:32 prog.c
-rw-rw-r-- 1 nikita nikita   19 сен 13 11:38 res.txt
nikita@nikita-pc:~/temp$ ls --sort=size -l
total 52
-rwxrwxr-x 1 nikita nikita 8656 сен 13 13:02 file_name
-rwxrwxr-x 1 nikita nikita 8656 сен 13 13:02 my_open
-rwxrwxr-x 1 nikita nikita 8608 сен 13 12:54 prog
-rw-rw-r-- 1 nikita nikita 1960 сен 13 12:45 my_open.o
-rw-rw-r-- 1 nikita nikita  397 сен 13 13:02 my_open.c
-rw-rw-r-- 1 nikita nikita   68 сен 13 11:32 prog.c
-rw-rw-r-- 1 nikita nikita   19 сен 13 11:38 res.txt
nikita@nikita-pc:~/temp$ ls -l | sort -k 5 -n
total 52
-rw-rw-r-- 1 nikita nikita   19 сен 13 11:38 res.txt
-rw-rw-r-- 1 nikita nikita   68 сен 13 11:32 prog.c
-rw-rw-r-- 1 nikita nikita  397 сен 13 13:02 my_open.c
-rw-rw-r-- 1 nikita nikita 1960 сен 13 12:45 my_open.o
-rwxrwxr-x 1 nikita nikita 8608 сен 13 12:54 prog
-rwxrwxr-x 1 nikita nikita 8656 сен 13 13:02 file_name
-rwxrwxr-x 1 nikita nikita 8656 сен 13 13:02 my_open
```

Фильтр Uniq

Изучим команду *Uniq*, которая читает исходный файл и сравнивает соседние строки. В обычном режиме вторая и последующие копии повторяющейся строки исключаются; остаток поступает в выходной файл, который не должен совпадать с исходным. Для этого выполним тестовое задание:

```
Выведите повторяющиеся размеры файлов из каталога /usr/bin. Для этого из вывода ls -l с помощью cut о
```

Результат фильтрации представлен в листинге:

```
nikita@nikita-pc:~/temp$ ls -l /usr/bin/ | cut -b24-31 | sort -n | uniq -d

       1
       2
       3
       4
       5
       6
       7
       8
       9
      10
      11
    ( ... )
     915
     1680
```

```
      1907
      4439
```

Фильтр Cmp

Изучим команду *Cmp*, которая производит побайтное сравнение и прекращает работу при первом несовпадении. Для этого выполним тестовое задание:

`Сравните файлы исходных текстов и объектные файлы созданных С-программ.`

Результат сравнения представлен в листинге:

```
nikita@nikita-pc:~/temp$ cat v1
9324893278309248903289
nikita@nikita-pc:~/temp$ cat v2
93F4893278309248903289
nikita@nikita-pc:~/temp$ cmp v1 v2
v1 v2 differ: byte 3, line 1
nikita@nikita-pc:~/temp$ cmp my_open.c prog.c
my_open.c prog.c differ: byte 1, line 1
nikita@nikita-pc:~/temp$ cmp my_open.o prog.o
my_open.o prog.o differ: byte 41, line 1
```

Фильтр Diff

Изучим команду *Diff*, которая выдает на стандартный вывод только те строки файлов, которые нужно изменить, чтобы привести файлы в соответствие друг с другом. Для этого выполним тестовое задание:

`Сравните prog.c и my_open.c с помощью diff:`
`diff -e prog.c my_open.c > eqv`

Результат сравнения представлен в листингах:

```
nikita@nikita-pc:~/temp$ cat v1
9324893278309248903289
9324893278309248903289
9324893278309248903289
9324893278309248903289
nikita@nikita-pc:~/temp$ cat v2
9324893278309248903289
93F4893278309248903289
9324893278309248903289
9324893278309248903289
nikita@nikita-pc:~/temp$ diff v1 v2
2c2
< 9324893278309248903289
---
> 93F4893278309248903289
nikita@nikita-pc:~/temp$ diff -e prog.c my_open.c > eqv
```

Результат выполнения команды *diff -e prog.c my_ open.c > eqv* автоматически сохраняется в файле *eqv*:

```
4c
if (argc > 1) {
      if (fopen(argv[1],"r") == NULL)
     printf("%s: неудача при попытке открыть файл %s \n",argv[0],argv[1]);
     else
     printf("%s: файл %s открыт \n",argv[0],argv[1]);
  }
  else
        printf("%s: открываемый файл не указан \n", argv[0]);
.
2c
main ( argc, argv )
  int argc;
  char *argv [];
.
```

Результатом выполнения программы являются строки, которые необходимо заменить, для того, чтобы файлы были одинаковые. Можно заметить, что *Diff* не посчитала, что нужно менять строки с "#include ", потому что они встречаются в обоих программах. Также команда выводит номер строки, начиная с которой должны быть произведены изменения.

1.3.7 Организация конвейера

Конвейер - некоторое множество процессов, для которых выполнено следующее перенаправление ввода-вывода: то, что выводит на поток стандартного вывода предыдущий процесс, попадает в поток стандартного ввода следующего процесса.

Рассмотрим различные методы организации конвейеров (где undefigned.txt это несуществующий файл, при открытии которого программа возвращает 1, а defigned.txt это существующий файл, при открытии которого программа возвращает 0):

```
nikita@nikita-pc:~/temp$ ./file_name undefigned.txt && ./file_name defigned.txt
./file_name: неудача при попытке открыть файл undefigned.txt
nikita@nikita-pc:~/temp$ ./file_name defigned.txt && ./file_name defigned.txt
./file_name: файл defigned.txt открыт
./file_name: файл defigned.txt открыт

nikita@nikita-pc:~/temp$ ./file_name undefigned.txt || ./file_name defigned.txt
./file_name: неудача при попытке открыть файл undefigned.txt
./file_name: файл defigned.txt открыт
nikita@nikita-pc:~/temp$ ./file_name defigned.txt || ./file_name defigned.txt ./file_name
    : файл defigned.txt открыт

nikita@nikita-pc:~/temp$ ./file_name undefigned.txt | ./file_name defigned.txt ./
    file_name: файл defigned.txt открыт
nikita@nikita-pc:~/temp$ ./file_name defigned.txt | ./file_name defigned.txt ./file_name:
     файл defigned.txt открыт

nikita@nikita-pc:~/temp$ ./file_name undefigned.txt ; ./file_name defigned.txt
./file_name: неудача при попытке открыть файл undefigned.txt
./file_name: файл defigned.txt открыт
nikita@nikita-pc:~/temp$ ./file_name defigned.txt ; ./file_name defigned.txt
./file_name: файл defigned.txt открыт
./file_name: файл defigned.txt открыт
```

Различия конвейеров:

- && - срабатывает только тогда, когда предыдущая команда вернула ноль.
- || - срабатывает только тогда, когда предыдущая команда вернула не ноль.
- | - стандартный выходной поток одной программы перенаправляется в стандартный входной поток другой программы.
- ; - исполняет команды друг за другом, ожидая окончания каждой из них перед началом выполнения другой.

Хорошим примером конвейера являются фильтры, рассмотренные в предыдущем пункте.