Перенаправление ввода/вывода

Некоторое время назад мы выполняли следующую задачу: искали telegram в выводе списка процессов и передавали утилите kill. Делать это было не очень просто, давайте оптимизируем процесс.

Во-первых, мы уже знаем команду *grep*, так что воспользуемся ею, чтобы найти процесс:

```
ps auxf | grep telegram-desktop
```

Но нам вернулись две строки, хотя telegram запущен только один. *grep* был запущен с аргументом в виде имени процесса, который мы искали, поэтому он нашёлся тоже.

Чтобы нашёлся только один наш процесс, нужно использовать ключ -v, который исключит строки со словом из выдачи.

```
ps auxf | grep telegram-desktop | grep -v grep
```

Вывод команды *ps* строго форматирован. Первая колонка — всегда имя пользователя, вторая — всегда PID. Такая идея довольно распространена в мире IT, ведь если информация структурирована, с ней гораздо проще работать.
Есть очень популярный формат CSV (Comma Separated Values). Это своего рода таблица — просто в текстовом виде, в которой ячейки разделяются запятой или переводом строки.

Из-за популярности структурирования информации в виде таблиц очень полезной нам будет **программа cut**, которая умеет выводить отдельные колонки таких документов. Она тоже умеет принимать различные аргументы. Два самых важных: -f, который позволяет выбирать нам номер колонки, и -d, позволяющий указывать делимитер — разделитель.

Давайте сделаем так, чтобы команда ps выводила только две колонки — PID и ту команду, которая запустила процесс. Для этого мы передадим названия колонок через повторяющийся аргумент -o:

```
ps -o pid -o cmd -xa
```

Теперь колонки разделены пробелом.

Далее нужно найти процесс telegram:

```
ps -o pid -o cmd -xa | grep telegram | grep -v grep
```

Теперь приходит время команды cut. Поскольку мы выводили только две колонки и PID в первой, нам потребуется аргумент -f1, а так как наш делимитер — пробел, зададим его при помощи -d " ".

```
ps -o pid -o cmd -xa | grep telegram | grep -v grep | cut -f1 -d " "
```

Теперь у нас есть PID!

Теперь нам требуется сделать так, чтобы это число стало аргументом для следующей команды — kill. Нам понадобится ещё одна команда — xargs. Она возьмёт наше число и покажет его команде kill как аргумент.

```
ps -o pid -o cmd -xa | grep telegram | grep -v grep | cut -f1 -d " " | xargs kill -9
```

Но мы убили telegram.

Разберём, что именно произошло на последнем шаге и почему нам потребовался xargs, хотя не требовался до этого.

Команда *cat*
Для этого сначала вспомним команду *cat.* Cat происходит от слова concatenate (объединять). Она может открыть нам файл и вывести его содержимое на консоль.

```
cat /etc/lsb-release
```

Она может объединять содержимое нескольких файлов.
Выглядит это примерно так:

```
cat file1 file2
```

Чтобы это стало полезным, мы можем при помощи символа >, оператора перенаправления, записать получившийся новый файл в другой, например:

```
cat file1 file2 > file3
```

Ещё при помощи этого же символа > мы можем отправить содержимое одного файла в другой, таким образом переписав файл. А символ >> позволит дописать новую информацию в конец файла.

```
cat file1 >> file 2
```

В Linux у каждой программы есть три стандартных потока:
STDIN — ввод.
STDOUT — вывод.
STDERR — ошибка.

Поток — это стандартизированный способ UNIX-систем оперировать данными, попадающими в программу. В определённом смысле их можно сравнить с потоками воды (например, как если бы вы включили кран на кухне) — они откуда-то берутся, куда-то попадают.

Поток ввода — всё, что попадает в программу, например с клавиатуры или через пайп от предыдущей отработавшей команды. **Поток вывода** — то, что команда выводит. Это может быть вывод на консоль. При помощи оператора перенаправления мы можем, например, записать вывод команды в файл.

У потоков есть нумерация: ввод — 0, вывод — 1, ошибки — 2.
Давайте посмотрим на примере:

```
cat /home/victoria/file
```

Поток STDERR — очень похож на поток вывода. В Linux их разделяют для удобства: мы можем выводить ошибки на консоль (поток вывода), чтобы сразу видеть проблему, или выводить их в файл (поток STDERR) — чтобы анализировать проблемы позже.

Выводы:

1. Можно или нет написать после очередного пайпа команду и ожидать, что она заберёт вывод предыдущего шага, зависит от того, умеет ли команда работать со стандартным вводом. Например, cat, grep умеют. Они принимают вывод предыдущей команды и отлично его обрабатывают. А вот команда kill — нет. Она может принять только значение аргумента. Для того чтобы стандартный ввод переформировать в вид аргумента, используется команда xargs.
2. >, >> — это перенаправление вывода (оператор перенаправления ввода, кстати, обозначается так же, только смотрит в другую сторону — <). То есть результат работы программы мы можем «забрать» с консоли и поместить в файл. Или забрать результат работы команды и поместить в файл.
3. Пайп — это перенаправление вывода на ввод. Когда мы говорили Ps auxf | grep telegram, мы буквально забрали со стандартного вывода результат выполнения команды ps и подали его на стандартный ввод команде grep (она, в отличие от kill, отлично работает со стандартным вводом).

Утилизация диска

Есть команда, которая позволяет оценить размер файлов. Это *du*. Лучше сразу использовать её с ключом -h, который будет показывать нам размерность, kB, MB, GB и так далее.

```
du -h hello_world.txt
```