# Linux Lab 7 - Parsing Text

Linux system admins worked from text-only terminals for years, so they developed some great utilities, tools, and methods for parsing text.

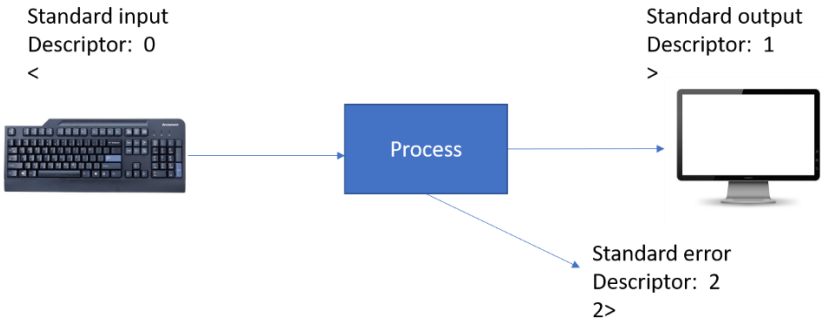
Note: although this lesson was written for a CentOS VM, everything should work equally well in Ubuntu.

## Read

"The Linux Command Line," chapter 6, Redirection, pp 49 -58.

Some points about the reading:

The OS has standard locations for sending and receiving output, the keyboard and the display.




These inputs and outputs may be **redirected**, so they go somewhere else; often it is the file system. The command from page 50 of the text

```
$ ls -l /usr/bin > ls-output.txt
```

redirects the output of the ls command from the display to a text file. The ">" character is the one that specifies redirection.

A **pipe** (or pipeline) makes the output of one command be the input of the next command. In a previous lab, the `ps aux` file gave us a lot of output and we were only looking for the line that contained "xlogo." So, we piped the output of the ps command into grep to search for lines containing xlogo.

```
$ ps aux | grep xlogo
```

## A Few Linux Text Utilities

We will use a file of data extracted from a Windows event log to practice using the Linux commands grep, cut, sort, and unique. This just scratches the surface of what is available, as there are entire books written on Regular Expressions (grep is named after them), sed, AWK, and other tools.

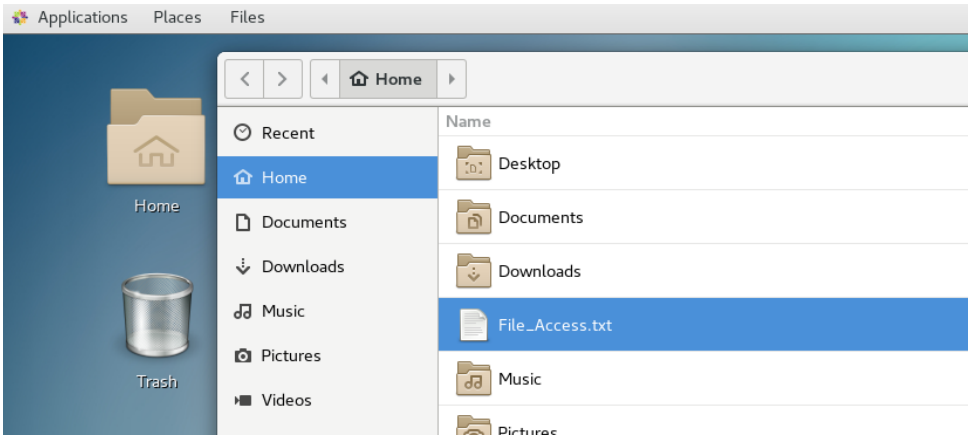
For this exercise we will pretend that Evil Hacker has broken into a Windows file share and stollen sensitive data. Fortunately, auditing for file access was active on the victim computer so we have a record of what the attacker did. Our job is to tell the organization what files the attacker accessed.

Part of the data in the file we have, File_Access.txt, includes the names of files that the attacker accessed. However, there is a lot of other data in this file, and the files accessed are listed multiple times. Our job is to extract just the file names. We could do this manually, but it would take a lot of

time to carefully examine the entire 1.6 MB+ file. Imagine the problem we would have if the file were 6 TB long! Enterprise log files can easily get that long. We can do our job quickly and easily with just a few commands.

## Download the test file

Download File_Access.txt from Canvas. If you've downloaded the file from your Windows Host (the OS you are running VMware Player on), you'll need to copy it to your Linux VM. If VMware Tools or open-vm-tools, is running on your VM, you'll be able to copy and paste the file directly into your Home directory as shown below.




If the tools are not working, a work-around would be to open Canvas from the Firefox browser on your Linux VM. If you do that, move the file from your Downloads directory up to your home ( ~, or /home/username ) directory.

From here on, everything we do will be in the terminal. (We could be cool and download the file from the command line, but it would be a lot of work to log in to Canvas and find the file we need.) Open a terminal and make sure you have File_Access.txt in your home directory.

```
john@ubuntu:~$ ls -l File_Access.txt
-rw------- 1 john john 1728094 Jun 26 16:01 File_Access.txt
```

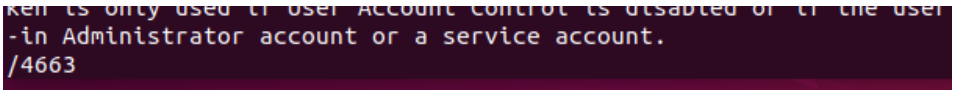
## Take a look at the file

First, we must do some reconnaissance. We need to examine the file to find a key phrase or character string we can search for so that we can extract the file names. The senior member of the incident response team has told us that file access on Windows systems is often recorded with Event ID 4663.

Open the file using less.

```
[john@john ~]$ less File_Access.txt
```

There is an easy to use search feature in less; type a forward slash (on the question mark key),enter the search term, and press return. In our case we are looking for 4663




```
Event[2914]:
  Log Name: Security
```

```
  Source: Microsoft-Windows-Security-Auditing
  Date: 2020-06-26T13:43:55.525
  Event ID: 4663
  Task: File System
  Level: Information
  Opcode: Info
  Keyword: Audit Success
  User: N/A
  User Name: N/A
  Computer: DESKTOP-UR71QBS
  Description:
An attempt was made to access an object.

Subject:
        Security ID:            S-1-5-21-999993552-654098596-4043940906-1002
        Account Name:           evilhacker
        Account Domain:         DESKTOP-UR71QBS
        Logon ID:               0x4E7A7

Object:
        Object Server:          Security
        Object Type:            File
        Object Name:            C:\SharedFiles\games\approvedgames.txt
        Handle ID:              0x2da8
        Resource Attributes:    S:AI

Process Information:
        Process ID:             0x4
        Process Name:

Access Request Information:
        Accesses:               READ_CONTROL

        Access Mask:            0x20000
```

It looks like all of the file names are preceded by "Object Name:". We can try that as a key phrase to search for.

## Extract lines containing our key phrase

In its most basic mode, grep extracts any line that contains our search term. Use "`grep --help`" to see the format that grep uses. For our use, you should find this command works well enough.

```
[john@john ~]$ grep "Object Name:" File_Access.txt
```

Whoa! Lots of text scrolls off the top of the screen. Pipe the results from grep into less so we have time to look at it.

```
[john@john ~]$ grep "Object Name:" File_Access.txt | less

        Object Name:    -
        Object Name:    -
        Object Name:    -
        Object Name:    -
        Object Name:            C:\SharedFiles\games\approvedgames.txt
        Object Name:            C:\SharedFiles\games\approvedgames.txt
        Object Name:            C:\SharedFiles\games
        Object Name:            C:\SharedFiles\CorporatePlans\productupgrades.xlsx
```

```
        Object Name:            C:\SharedFiles\CorporatePlans\productupgrades.xlsx
        Object Name:            C:\SharedFiles\CorporatePlans\productupgrades.xlsx
        Object Name:            C:\SharedFiles\CorporatePlans\productupgrades.xlsx
<snip>
```

Save the output from the last command to a file using redirection.

```
[john@john ~]$ grep "Object Name:" File_Access.txt > file1
```

## Cut out unneeded columns

The cut command allows us to select columns of data, as we could if the data were in a spreadsheet. In our case, the data we want is in the last column. We want to get the last column by itself so we can work on it.

Enter "`cut --help`" to see how the cut command works. The options we can use here are, -d [delimiter], and -f [field number, staring at 1.] The delimiter is the character that separates the columns. In our case, it could be a space or a tab, or possibly a colon. Note: if you don't specify a delimiter, cut uses a tab. Also note that you can cut multiple columns (-f 2,3 grabs columns 2 and 3.) We only need the last column right now.

It is often difficult to tell spaces and tabs apart in printed output. You need to know where the spaces and tabs are in your output if you are to choose your delimiter well. One way to do this is to use
`cat -A file1.txt | less`
If you do that, tabs will show up as `^I` and line feeds (Linux end of line) will show up as `^M`. (Note: the carriage return that Windows uses will be `^R`.)

```
^IObject Name:^I-^M$
^IObject Name:^I-^M$
^IObject Name:^I^IC:\SharedFiles\games\approvedgames.txt^M$
^IObject Name:^I^IC:\SharedFiles\games\approvedgames.txt^M$
^IObject Name:^I^IC:\SharedFiles\games^M$
^IObject Name:^I^IC:\SharedFiles\CorporatePlans\productupgrades.xlsx^M$
^IObject Name:^I^IC:\SharedFiles\CorporatePlans\productupgrades.xlsx^M$
```

Now you can see that you should probably use tab as your delimiter, since the file name you want always has a tab (`^I`) just before it. Again, to use tab as a delimiter, just leave the -d option out. Your command should look something like this, except you need to determine what field number to use. If you get no output, just experiment with the field number.
`cut -f {field number goes here} file1.txt`

Pipe the output into less so you can scroll through it if you want to. If you are successful, the top of the output will look like this:

```
{some blank lines}

C:\SharedFiles\games\approvedgames.txt
C:\SharedFiles\games\approvedgames.txt
C:\SharedFiles\games
C:\SharedFiles\CorporatePlans\productupgrades.xlsx
C:\SharedFiles\CorporatePlans\productupgrades.xlsx
```

Once you have what you want, redirect the output into file2.

## One file, one entry

The data is full of duplicates, and we would like to remove them. The uniq command does that well, but it only works on data that has been sorted. Therefore, we use the sort command first.

```
[john@john ~]$ sort file2.txt | less
```

Here is sample output, scrolled down so it shows something other than "\".

```
{some blank lines}
C:\SharedFiles
C:\SharedFiles\Accounting
C:\SharedFiles\Accounting
C:\SharedFiles\Accounting\CEOobjectives.docx
C:\SharedFiles\Accounting\CEOobjectives.docx
C:\SharedFiles\Accounting\CEOobjectives.docx
C:\SharedFiles\Accounting\DisciplineRecords.docx
```

We can pipe the sorted output into the uniq command

```
[john@john ~]$ sort file2.txt | uniq | less
```

```
{some blank lines}
C:\SharedFiles
C:\SharedFiles\Accounting
C:\SharedFiles\Accounting\CEOobjectives.docx
C:\SharedFiles\Accounting\DisciplineRecords.docx
C:\SharedFiles\Accounting\EmployeeSalaries.xlsx
```

Voila! We could keep going to clean up the list some more, but this demonstrates the point. Note: we could have used `sort -u` instead of piping the results of sort into `uniq`.

## Putting it all together

A true Linux sysadmin does not usually create intermediate files like we did. Instead, they will put it in one command like this:

```
[john@john ~]$ grep "Object Name:" File_Access.txt | cut -f 4 | sort | uniq > answer.txt
```

Note that the command redirects the output to a file called answer.txt. Hand in that file using Canvas to show you did this lab.

A good sysadmin may test a new command one step at a time, however, piping the results of each step into less for easier troubleshooting. For example, a sysadmin may follow this sequence:

`grep "Object Name:" File_Access.txt | less` (looks at data, and fixes errors if we grabbed the wrong thing.)

`grep "Object Name:" File_Access.txt | cut -f 4 | less` (examines data, fixes errors)

`grep "Object Name:" File_Access.txt | cut -f 4 | sort | uniq | less` (examines data, fixes errors)

`grep "Object Name:" File_Access.txt | cut -f 4 | sort | uniq > answer.txt`

## Hand in

Submit a copy of your answer file.