System Programming - Linux
Week 8 - Input / Output Redirection

Lecturer: Dr. Aggrey Obbo (PhD)

May 30, 2025

Contents

1 Introduction

The presentation on Input/Output (I/O) redirection in Linux helps the reader grasp how to manage the primary data streams of input, output, and error which is essential for excelling at the command line. The notes will provide understanding on the knowledge to redirect these streams to and from files, as well as to link commands together using pipes. By the end of the module, one will be capable to direct the flow of data, facilitating powerful and efficient command-line operations for a variety of tasks.

Objectives

- Define stdin, stdout, stderr and their default file descriptors.
- Explain the purpose and benefits of I/O redirection and piping in Linux.
- Perform output redirection (overwrite and append) for stdout and stderr, individually and combined.
- Perform input redirection from files, here documents, and here strings.
- Construct and explain command pipelines using the — operator.
- Apply I/O redirection and piping to solve practical command-line tasks and understand best practices.

2 Standard Data Streams

[1][2] A stream is a sequence of bytes that can be either read or written. Standard streams are pre-connected input and output communication channels between a computer program and its environment when it begins execution.They offer an interface for transferring and processing data across different input/output tasks and it is the basic idea for managing input, output, and interaction between processes.

Linux commands interact with three fundamental data streams. These are: Standard Input (stdin), Standard Output (stdout), and Standard Error (stderr). And by default, all these are connected to the terminal.

- Standard Input (stdin)[0]: The standard input stream refers to the information entered via the keyboard at the system console. This input stream can be redirected, allowing the content of a file to be brought into a program or command. Example: The cat command, when used without a filename, reads from stdin which is the keyboard. Typing text followed by Enter, and then Ctrl+D (EOF), sends that input to cat, which then echoes it to stdout:

  cat
  Input from keyboard!
  [Press Enter]
  This is the exact input from the keyboard.
  [Press Enter]
  ctrl+D # To exit terminal

  Alternatively, one can use the operator < to redirect a file's content to a command's stdin:

  echo "Optimized for learners" < TutorialsArc.txt grep "learners"< input.txt Optimized for Learners grep receives search input from TutorialsArc.txt via stdin.

- Standard Output (stdout)[1]: The standard output stream (stdout) serves as the default destination for the regular output produced by commands. By default, stdout is linked to the terminal screen, from where the user observes output after execution of a command.

  stdout can be redirected to a file using the ">" operator. This allows saving the command's output for future reference instead of displaying it on the screen. This redirection is fundamental for managing and processing data within the Linux environment.

  Example:

  ls -l lists files and directories, sending the details to stdout: ls -l total 2160 -rw-rw-r– 1 study study 42 May 7 08:04 datastreams.txt -rw-rw-r– 1 study study 26 May 7 08:14 datastream.txt

  But this can be redirected as follows:

  ls -l > TutorialsArc.txt
  And cat TutorialsArc.txt displays content of the TutorialsArc.txt
  total 2160
  -rw-rw-r– 1 study study 42 May 7 08:04 datastreams.txt
  -rw-rw-r– 1 study study 26 May 7 08:14 datastream.txt
  .
  .
  .

- Standard Error (stderr)[2]: Standard error stream (stderr) is a distinct output channel dedicated to error messages, warnings, and diagnostic information produced by commands. It helps to separate error messages from standard output (stdout). By default, stderr is linked to the terminal display.

  Standard error stream can be redirected to a file using the 2> operator, which enables the capture and examination of error messages separately. This is helpful for debugging scripts and handling automated processes.

  Example:

Trying to access a non-existent file with cat, sends an error message to stderr

cat missingfile.txt
cat: missingfile.txt: No such file or directory
2> operator redirects stderr to a specified file - Errorlog.txt
cat missingfile.txt 2> Errolog.txt
cat Errorlog.txt
cat: missingfile.txt: No such file or directory

These three standard streams Standard Input (stdin), Standard Output (stdout), and Standard Error (stderr) with respective file descriptors: 0 (stdin), 1 (stdout), 2 (stderr) respectively are fundamental for how commands interact with data and provide feedback to the user.

3 Redirection

In Linux, redirection lets one modify the standard input, output, and error streams of commands. We can change the source of input to a file (<), while directing the output (>) or appending output (>>) and errors (2>, 2>>) to files. This feature facilitates dynamic data management and automation by managing the transfer of information to and from commands.

3.1 Output Redirection

Output redirection is a feature that enables altering the default destination of a command's standard output (stdout). Output can be sent to a file or be "piped" to another command. This helps to save command results for further analysis or pass the output of one command as input of another.

The operator for basic output redirection is the ">" symbol. The syntax has > followed by a filename. If the file doesn't exist, it is created. If the file already exists, its contents are overwritten.

Example
ls -l > file_listing.txt
The output of the above will be saved to the file file_listing.txt instead of the default monitor. If the file exists, it will be overwritten. The file can then be viewed using the cat command as follows:

cat file_list.txt

On the other hand, to append the output to an existing file instead of overwriting it, use the >> operator as follows:

date >> file_listing.txt This will append output of the date command to the end of the file_listing.txt file, preserving its previous contents.

This is useful for adding information to log files or accumulating output from multiple commands. Output redirection is a cornerstone of shell scripting and command-line efficiency in Linux, enabling you to manage and process data flow effectively. It allows you to capture the results of commands for later analysis, use them as input for other tools, and automate tasks by controlling where command output is directed. Remember the crucial difference between > (overwrite) and >> (append) to avoid unintended data loss.

3.2 Input Redirection

Input redirection allows changing the default source of a command's standard input (stdin) to receiving input to read data from a file or any other source. This is essential for data processing and automation of tasks.

The syntax has < followed by a filename from where the input should be expected. For example;

wc -l command is used to counts the number of lines in the input it receives. And the file in this case will be TutorialsArc.txt
wc -l < TutorialsArc.txt

The "here document", denoted by ¡¡ allows to embed multiple lines of input directly within your script or command.

We specify a delimiter after ¡¡, and all subsequent lines until that same delimiter is encountered are treated as stdin for the command. For example: cat command normally displays content of files. The here document enables providing multi-line text as input to cat:

cat << STOP_INPUT
TutorialsArc Optimized for Learners - line 1
TutorialsArc Optimized for Learners - line 2
Last line provided by the here document
STOP_INPUT

The delimiter STOP_INPUT is arbitrary; you can choose any unique string.

The "here strings" (<<<) on the other hand provides a way to redirect a single line of input to a command. For instance, the grep command searches for a pattern in its input. The here string can be used to search for a word within a short string:

grep "TutorialsArc" <<< "This line contains the word TutorialsArc."

3.3 Error Redirection

The standard error stream (stderr) is meant to output error messages, warnings, and diagnostic information, to the standard output. By default, stderr is also directed to the terminal screen. However, Linux provides a means of redirecting the stderr. To redirect stderr, the file descriptor number 2 followed by the redirection operator >. Like the stdout, if the file doesn't exist, it's created, and if it exists, its overwritten.

cat tutor.txt where tutor.txt is a non existent file.
The output should be as follows:
cat: tutor.txt: No such file or directory
The error message can be saved to a file named Errorlog.txt, as follows:
cat tutor 2> Errorlog.txt
To view the error log, use
cat error_log.txt.

And to append stderr to an existing file use the 2>> operator as follows;
cat tutor.txt 2>> Errorlog.txt
This is useful for accumulating error logs over time without losing previous error messages.

Stdout and stderr are normally to the same file. And this can be done in the following ways. A common requirement is to redirect both stdout and stderr to the same file. This can be done in the following ways:

- To first redirect stdout to the file (¿ filename) and then redirect stderr to the same location as stdout using 2>&1. The order is crucial. For example;
  command > combined_log.txt 2>&1

- Alternatively, shells such as Bash provide a more concise syntax &> or >& to redirect both stdout and stderr to the specified file:
  command &> combined_log.txt

Error redirection is invaluable when debugging scripts and logging errors from automated tasks. They can also be used to separate successful output from diagnostic information, making it easier to identify and resolve issues.

Redirection and piping, forms the backbone of flexible and efficient command-line operations in Linux, enabling us to manipulate data flow in various ways.

4 Pipes (|)

A pipe is a mechanism for redirecting output to another destination for further processing. In Linux, it enables Inter-Process Communication (IPC) that allows to connect the standard output (stdout) of one command directly to the standard input (stdin) of another command. They enable complex processing that combine two or more commands or programs [3][4].

The syntax for using a pipe is as follows:
The pipe symbol | is placed between the two communicating commands. The output of the command on the left side

of the pipe is fed as input of the command on the right side. The shell then executes the commands concurrently. An example is as follows:

ls -l | grep "Optimized" Here, the ls -l command generates a list of files and directories from its stdout. This output is then piped directly as input to the grep "Optimized" command. The grep utility then filters this input, displaying only the lines that contain the string "Optimized". The final result, the list of files containing "Optimized", is then displayed on the terminal which is the standard output.

Pipes can also be joined to create more complex data processing pipelines. For example;

ls *.txt | sort |head -n 10

ls *.txt lists all text files, sort the result alphabetically, and then displays the first 10 lines.

The beauty of pipes lies in their ability to combine simple, specialized tools to perform intricate tasks without the need for intermediate files. The data flow directly from one command to the next in memory, without the need for any intermediate files, making the process efficient.

Pipes are a basis for many command-line productivity and shell scripting in Linux.

5 Examples on Pipes

To effectively process large data files in Linux requires a combination of grep, awk, and sed command-line tools. These utilities enable the management of text-based data without the need to load the complete file into memory at once.

Global Regular Expression Print (grep) is used for filtering lines in a file that match a specific pattern or regular expression. It is invaluable for extracting relevant information from large datasets.

grep Examples

Example 1:
grep "192.168.1.100" Errolog.log
This command will efficiently scan Errorlog.log and echo every line containing the string "192.168.1.100".

Example 2: Excluding lines containing a specific pattern:

To find all log entries that do not contain the word "optimized":

grep -v “optimized” Errorlog.log

AWK Examples

The awk utility is used in processing columns and performing actions. awk can perform actions such as calculations, conditional logic and printing.
Example 1:Extracting columns:
awk -F ',' 'print 1,2' tutorialsarc.txt
-F defines the delimiter print $1,$2 tells awk to print the first field ($1), and then the second field ($2).
Example 2: Performing calculations
awk '-F",' 'print $2 / ($3*$3)' tutorialsarc.txt

This awk script divides column 2 by the square of column 3 for each line.

SED Examples

The SED utility is a stream editor used for transforming text. It processes text line by line, applying specified commands like substitution, deletion, and insertion without requiring interactive input.

Stream EDitor (SED) is a Linux utility used for performing basic text transformations on an input stream. It is often used for substitutions, deletions, and insertions.

Example 1: Replacing a string:

SED allows replacement of an old string with a new string on a large file.

sed 's/oldstring/newstring/g' document.txt

The s command performs substitution, /old/new/ specifies the pattern to replace and the replacement string, g indicates that all occurrences on a line should be replaced.

Example 2: Deleting line(s) based on a pattern:

To remove all lines containing the word "Optimized" from a file (Errorlog.txt):

sed '/Optimized/d' Errorlog.txt The /Optimized/ part specifies the pattern to match, and d is the delete command.

Combining any of grep, awk, and sed in conjunction with pipes can enable robust and efficient workflows.

6 Conclusion

In conclusion, input/output redirection is an essential feature in Linux that greatly improves command-line flexibility. By using operators such as <, >, and >> to manipulate standard streams, we can manage data flow, directing input from files and piping output to various commands.

This functionality simplifies intricate tasks, facilitating effective data management, automation, and scripting. Grasping the concept of redirection is vital for becoming proficient in the Linux environment, as it enables users to construct powerful command sequences and handle information accurately. Whether dealing with extensive datasets or automating system activities, I/O redirection is a crucial asset.

List of References

1. Shotts, W. (2017). The Linux command line. LinuxCommand. org.
2. Stevens, W. R., Rago, S. A. (2013). Advanced programming in the UNIX environment. Addison-Wesley.
3. Negus, C. (2012). Linux bible (Vol. 772). John Wiley Sons.
4. Robbins, A. (2005). Unix in a Nutshell. " O'Reilly Media, Inc.".

Appendix - Some Redirection Example Codes

Use the cat command to view standard input

- First, create a text file named tutotialsarc.txt
  echo "My redirect text!" > tutotialsarc.txt
- Use < to redirect the content and cat to read the input
  cat < tutotialsarc.txt
- Output is:
  My redirect text!

Use the grep command to search for patterns in a text.

- If textfile is tutotialsarc.txt and contains the content:
  Atim:user
  Mosh:admin
  Kim:user
  Joan:superadmin
  Peter:admin
- If we need all lines containing "admin", use as follows:
  grep "admin" < tutotialsarc.txt

- Output:
  Mosh:admin
  Joan:superadmin
  Peter:admin

The same principles apply for Output Redirection (>). Using ls with output redirection

- Redirect ls -l out to text file tutorialsarc.txt
  ls -l > tutorialsarc.txt

- Use cat to display content of tutorialsarc.txt

For Error Redirection (2>)

- Trying to cat a non-existent file
- cat ugandas 2> tutorialsarc.log - This redirects the error to a file named tutorialsarc.log
- View the error message: cat tutorialsarc.log
- Output of tutorialsarc.log

- Output is: cat: ugandas: No such file or directory

Exercise: Try output redirection using > and >>. What difference do you observe?