Tutorial5: Redirection

Contents

REDIRECTION: STANDARD INPUT / STANDARD OUTPUT / STANDARD ERROR

Main Objectives of this Practice Tutorial

- Understand and use the **cut**, **tr**, and **wc** Linux commands
- Define the terms **Standard Input** (*stdin*), **Standard Output** (*stdout*), and **Standard Error** (*stderr*)
- Understand and use the **>**, **>>**, **2>**, **2>>** symbols with Linux commands
- Understand the purpose of the the **/dev/null** file and the **Here Document**
- Define the term **pipeline command** and explain how a pipeline command functions
- Define the term **filter** and how it relates to pipeline commands
- Use the **semicolon** ";" and **grouping** "( )" symbols to issue multiple Unix / Linux commands on a single line
- Use the **backslash** "\" symbol to spread-out long Unix/Linux commands over multiple lines

Tutorial Reference Material

Course Notes	Linux Command / Shortcut Reference		YouTube Videos
Slides:	**Redirection:**	**Redirection Filters:**	**Brauer Instructional Videos:**

- Week 5 Lecture 1 Notes:
  PDF (https://wiki.cdot.senecacollege.ca/uli101/slides/ULI101-5.1.pdf) | https://wiki.cdot.senecacollege.ca/uli101/slides/ULI101-5.1.pptx PPTX]
- Week 5 Lecture 2 Notes:
  PDF (https://wiki.cdot.senecacollege.ca/uli101/slides/ULI101-5.2.pdf) | PPTX (https://wiki.cdot.senecacollege.ca/uli101/slides/ULI101-5.2.pptx)

- Standard Input (stdin) (http://www.linfo.org/standard_input.html)
- Standard Output (stdout) (http://www.linfo.org/standard_output.html)
- Standard Error (stderr) (http://www.linfo.org/standard_error.html)
- Pipeline Commands (http://www.linfo.org/pipe.html)

Multiple Commands:

- Semicolon (https://www.javatpoint.com/linux-semicolon)
- Grouping ( ) (https://www.gnu.org/software/bash/manual/html_node/Command-Grouping.html)

- more (http://man7.org/linux/man-pages/man1/more.1.html) , less (http://man7.org/linux/man-pages/man1/less.1.html)
- head (http://man7.org/linux/man-pages/man1/head.1.html) , tail (http://man7.org/linux/man-pages/man1/tail.1.html)
- sort (http://man7.org/linux/man-pages/man1/sort.1.html)
- uniq (http://man7.org/lin

- Reading/Writing to Files (echo, stdin, stdout, stderr, >, >>, 2>, cat, more, less, man, date, diff, diff -y, find, wc (https://www.youtube.com/watch?v=ocU34PcYn2U&list=PLU1b1f-2Oe90TuYfifnWulINjMv_Wr16N&index=4))

ux/man-pages/man1/uniq.1.html)

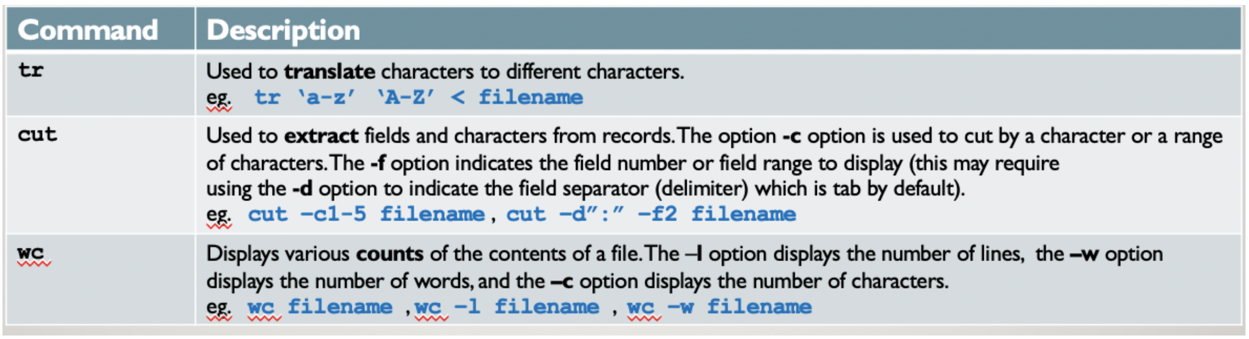
- grep (http://linuxcommand.org/lc3_man_pages/grep1.html)
- cut (http://man7.org/linux/man-pages/man1/cut.1.html)
- tr (http://linuxcommand.org/lc3_man_pages/tr1.html)
- wc (http://man7.org/linux/man-pages/man1/wc.1.html)
- tee (http://man7.org/linux/man-pages/man1/tee.1.html)

KEY CONCEPTS

Additional File Manipulation Commands

Before proceeding, let's look at some additional commands used to manipulate content of text files.

Refer to the table below regarding these text file manipulation commands:



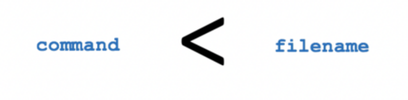
Command	Description
tr	Used to **translate** characters to different characters. eg. `tr 'a-z' 'A-Z' < filename`
cut	Used to **extract** fields and characters from records. The option **-c** option is used to cut by a character or a range of characters. The **-f** option indicates the field number or field range to display (this may require using the **-d** option to indicate the field separator (delimiter) which is tab by default). eg. `cut -c1-5 filename` , `cut -d":" -f2 filename`
wc	Displays various **counts** of the contents of a file. The **–l** option displays the number of lines, the **–w** option displays the number of words, and the **–c** option displays the number of characters. eg. `wc filename` , `wc -l filename` , `wc -w filename`

Redirection (Standard Input, Standard Output, Standard Error)

***Redirection** can be defined as changing the way from where commands read input to where commands sends output. You can redirect input and output of a command.*

Reference: https://www.javatpoint.com/linux-input-output-redirection

Standard input (**stdin**) is a term which describes from where a command receives **input**.
This would apply only to Unix/Linux commands that accept stdin input (like *cat*, *more*, *less*, *sort*, *grep*, *head*, *tail*, *tr*, *cut*, *wc*, etc.).

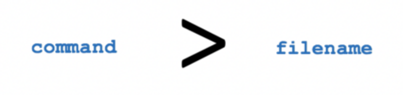


The **standard input** (**stdin**) symbol that describes where a Unix/Linux command receives **input**

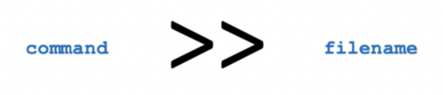
Examples:

```
tr 'a-z' 'A-Z' < words.txt
cat < abc.txt
sort < xyz.txt
```

Standard output (**stdout**) describes where a command sends its **output**.
In the examples below, output from a command is sent to the **monitor**, unless it is sent to a **text file**.



The **standard out** (**stdout**) symbol with one greater than sign **overwrites** existing file content with command output

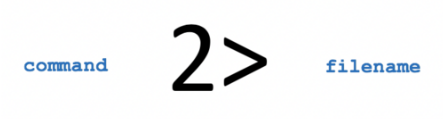


The **standard output** (**stdout**) symbol with two greater than signs **add** command's output to **bottom** of existing file's contents.

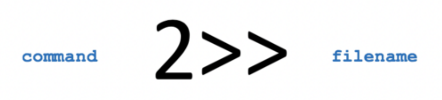
Examples:

```
ls -l
ls -l > detailed-
listing.txt
ls /bin >> output.txt
```

Standard Error (**stderr**) describes where a command sends it's error messages. In the examples below we issue the pwd in capitals on purpose to generate an error message, which can be redirected to a **text file**.



The **standard error** (**sterr**) symbol with one greater than sign **overwrites** existing file content with command's **error message**.



The **standard error** (**stderr**) symbol with two greater than signs **add** command's error message to **bottom** of existing file's contents.

Examples:

```
PWD
PWD 2> error-message.txt
PWD 2 >> error-messages.txt
PWD 2> /dev/null
```

The /dev/null File

The **/dev/null** file (sometimes called the **bit bucket** or **black hole**) is a special system file that **discard** all data written into it. This is useful to discard unwanted command output.

Examples:

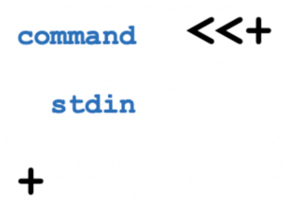
```
LS 2> /dev/null
ls > /dev/null
find / -name "tempfile" 2> /dev/null
```

The Here Document

In Linux, the **Here Document** allows a user to redirect stdin from within the command itself.

Example:

```
cat <<+
Line 1
Line 2
Line 3
+
```



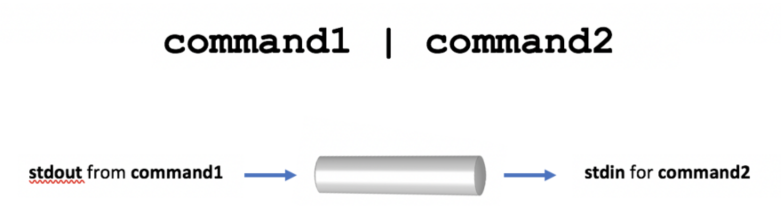
The **Here Document** allows a user to redirect stdin from within the command itself.

Pipeline Commands

Pipeline Command: Having commands send their **standard output** directly to **standard input** of other commands WITHOUT having to use **temporary** files.

Pipes that are used in a **pipeline command** are represented by the **pipe** "|" symbol.

A few simple commands can be combined to form a more powerful command line.



A **pipeline command** sends a command's **standard output** directly to **standard input** of other command(s) without having to create temporary files.

Commands to the **right** of the pipe symbol are referred to as **filters**. They are referred to as *filters* since those commands are used to **modify** the stdout of the previous command. Many commands can be "piped" together, but these commands (filters) must be chained in a specific order, depending on what you wish to accomplish

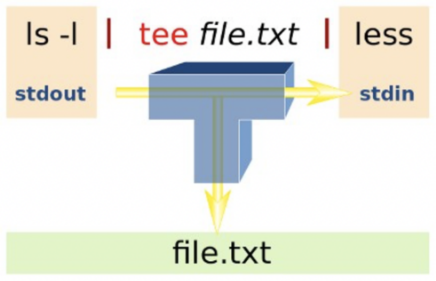
Examples:
```
ls -al | more
ls | sort -r
ls | sort | more
ls -l | cut -d" " -f2 | tr 'a-z' 'A-z"
ls | grep Linux | head -5
head -7 filename | tail -2
```

The tee Command

The **tee** utility can be used to split the flow of **standard output** between a **text file** and the **terminal screen**.

The **tee** option **-a** can be used to add content to the **bottom** of an existing file as opposed to *overwriting* the file's previous contents.

The reason for the name "**tee**" is that the splitting of the flow of information resembles a capital T.



The **tee** utility can be used to **split** the flow of information. For example to save in a file as well as display on a screen.
(Image licensed under cc (https://creativecommons.org/licenses/by-sa/3.0/))

Examples:
```
ls | tee unsorted.txt | sort
ls | grep Linux | tee matched.txt | more
ls | head -5 | tee -a listing.txt
```

Multiple Commands Using Semicolon, Grouping, and Backquotes

Besides piping, there are other ways that multiple commands may be placed in one line: commands may be separated by **semi-colons**.

Example:

```
sleep 5; ls
```

Multiple commands can also be **grouped** by using parentheses.

Example:

```
(echo "Who is on:"; w) > whoson
```
(***Note:*** *all command output is sent to a file*)

Commands may also be **spread-out over multiple lines**, making it easier (for humans) to interpret a long command.

The \ symbol "*quotes-out*" the meaning of the **ENTER** key as text (i.e. *new-line* as instead of *running* the command).

Example:

```
echo "This will be split over multiple \
lines. Note that the shell will realize \
that a pipe requires another command, so \
it will automatically go to the next line" |tr '[a-z]' '[A-Z]'
```

INVESTIGATION 1: BASICS OF REDIRECTION

ATTENTION: This online tutorial will be required to be completed by **Friday in week 6 by midnight** to obtain a grade of **2%** towards this course

In this investigation, you will learn how to redirect **standard input**, **standard output** and **standard error** when issuing Unix / Linux commands.

Perform the Following Steps:

1. **Login** to your matrix account and issue a command to **confirm** you are located in your **home** directory.
2. Issue the following Linux command to create the following directory: `mkdir ~/redirect`
3. Change to the **~/redirect** directory and confirm that you changed to that directory.
4. Use a text editor to create a file in your current directory called **data.txt** and enter the following text displayed below:

   ```
   This is line 1
   This is line 2
   This is line 3
   ```

5. **Save** editing changes and **exit** the text editor.
6. Issue the following Linux command: `tr 'a-z' 'A-Z' < data.txt`

   What does this command do?

7. Issue the following Linux command: `tr 'a-z' 'A-Z' < data.txt > output.txt`

   What does this command do? What are the contents of the file *output.txt*?

8. Issue the following Linux command: `tr 'a-z' 'A-Z' > output.txt < data.txt`

   What does this command do? Is there any difference in terms of this command and the previous command issued?

9. Issue the following Linux command: `tr 'a-z' 'A-Z' >> output.txt < data.txt`

   What happens to the content of the **output.txt** file? Why?

10. Issue the following Linux command: `tail -2 < data.txt > output.txt`

    What does this command do? Check the contents of the **output.txt** file to confirm.

11. Issue the following Linux command: `tail -2 > output2.txt < data.txt`

    Why does this command render the same results as the previous command?
    Try explaining how the command works in terms of **stdin** and then **stdout**.

12. Issue the following Linux command to create a file: `cat > output3.txt`

13. Enter the follow text displayed below:

    ```
    This is the file output3.txt
    ```

14. Press `ctrl-d` to exit the command.

15. Issue the **cat** command to view the contents of the file: **output3.txt**

16. Issue the following Linux command: `cp ~jason.carman/uli101/cars .`

17. Issue the **cat** command to view the contents of the **cars** file.

18. Issue the following Linux command: `cut -c1-10 cars`

    What did this command do?

19. Issue the following Linux command: `cut -f5 cars > field5.txt`

    What did this command do?
    Check the contents in the file **field5.txt** to see what happened.

20. Issue the following Linux command: `cut -f1-3 cars > field123.txt`

    What did this command do? (check file contents)

21. Issue the following Linux command: `cut -f1,5 cars > field15.txt`

    What did this command do? (check file contents)

22. Issue the following Linux command: `wc cars > count.txt`

    What information does the **count.txt** file contain?

23. Issue the following Linux command: `wc -l cars > count1.txt`

    What information does the **count1.txt** file contain?

24. Issue the following Linux command: `wc -w cars > count2.txt`

    What information does the **count2.txt** file contain?

25. Issue the following Linux command: `ls -l > listing.txt`

    What information does the **listing.txt** file contain?

26. Issue the following Linux command: `pwd > listing.txt`

    What happenned to the original contents of the file called **listing.txt**? Why?

27. Issue the following Linux command (use 2 greater-than signs): `date >> listing.txt`

    What information does the **listing.txt** file contain? Why?

28. Issue the following Linux command: `cat listing.txt cars > combined.txt`

    What information does the **combined.txt** file contain? Why?

    NOTE: The **cat** command stands for "**concatenate**" which means to **combine** contents of multiple files into a single file.
    This is why the command is called "*cat*".

29. Issue the following Linux command: `cat listing.txt cars murray 2> result.txt`

    What is displayed on the monitor? What information does the **result.txt** file contain? Why?

30. Issue the following Linux command: `cat listing.txt cars murray > myoutput.txt 2> /dev/null`

    What is displayed on the monitor? What happened to the error message?

31. Issue the following Linux command: `cat listing.txt cars murray > myoutput.txt 2> result.txt`

    What is displayed on the monitor? what do those files contain? Why?

    The **Here Document** allows you to redirect stdin from with the Linux command itself. Let's get some practice using the Here Document.

32. Issue the following Linux command:
    ```
    cat <<+
    line 1
    line 2
    line 3
    +
    ```

    What do you notice?

33. Issue the following Linux command:
    ```
    grep 2 <<+
    line 1
    line 2
    line 3
    +
    ```

    What do you notice? How does this differ from the previous command? Why?

34. Issue the following Linux command:
    ```
    grep 2 > line2.txt <<+
    line 1
    line 2
    line 3
    +
    ```

    What do you notice? What is contained in the file **line2.txt**? Why?

    NOTE: You will now run a shell script to confirm that you properly issued Linux commands using redirection.

35. Issue the following Linux command to run a checking script:
    `~uli101/week5-check-1`

36. If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

37. Issue the **ls** command to see all of the **temporary files** that were created as a result of redirection.

    The problem with using these redirection symbols is that you create **temporary text files** that take up **space** on your file system.

38. Issue a Linux command (using **Filename Expansion**) to **remove** those temporary text files in the current directory.

39. Issue the following Linux command to check that you removed ALL of those temporary text files:
    `~uli101/week5-check-2`

40. If you encounter errors, make corrections and **re-run** the checking script until you receive a congratulations message, then you can proceed.

    In the next investigation, you will be learning how to issue **pipeline Linux commands** which can accomplish tasks without creating temporary files.

INVESTIGATION 2: REDIRECTION USING PIPELINE COMMANDS

In this investigation, you will learn to issue **pipeline commands** to to accomplish tasks without having to generate temporary files.

Perform the Following Steps:

1. Confirm that you are still located in the **~/redirect** directory.

   The **problem** with creating temporary files, is that they take up space on your server, and should be removed. You actually did that in the previous investigation.

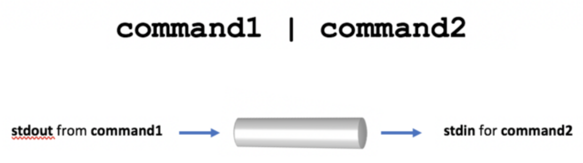
   You will be issuing a **pipeline command** which will use the pipe symbol "|" that will send the stdout from a command as stdin into another command without having to create temporary files.

2. Issue the follow Linux **pipeline command**: `ls /bin | more`

   What happened? Press **q** to exit display.

3. Issue the following Linux **pipeline command**: `ls /bin | who`

   What happened? Although this pipeline command provides output, it **does not work** properly as a pipeline command since the **who** command is **NOT** designed to accept standard input.

   NOTE: When issuing pipeline commands, commands to the right of the pipe symbol must be designed to accept **standard input**. Since the *who* command does not, you did NOT see the contents of the **/bin** directory but only information relating to the *who* command. Therefore, the **order** of which you build your pipeline command and the **type of command** that is used as a *filter* is extremely important!

4. Issue the following Linux command: `ls /bin/?? > listing.txt`

5. Issue the following Linux command: `sort listing.txt`

6. Issue the following Linux command to remove the listing file: `rm listing.txt`

7. Issue the following Linux **pipeline command**: `ls /bin/?? | sort`

   You should notice that the output from this pipeline command is the same output from the command you issued in **step #5**.

8. Issue the following Linux **pipeline command**: `ls /bin/?? | sort | more`

What is difference with this pipeline command as opposed to the previous pipeline command? Press **q** to exit display.

9. Issue the **ls** command.

   You should notice that **no files have been created**.
   Let's get practice issuing more pipeline commands using commands
   (previously learned or new) to be used as **filters**.

10. Issue the following Linux **pipeline command**: `ls /bin/?? | sort | head -5`

    What did you notice?

11. Issue the following Linux **pipeline command**: `ls /bin/???? | sort | grep r | tail -2`

    What did you notice? Could you predict the output prior to issuing this pipeline command?

12. Issue the following Linux **pipeline command**: `ls /bin/???? | sort | grep r | cut -c1-6`

    Try to explain step-by-step each process in the pipeline command (including *filters*)
    to explain the final output from this pipeine command.

13. Confirm that you are still located in the **~/redirect** directory.

14. Issue the following Linux **pipeline command**:
    `ls /bin/???? | tee unsort.txt | sort | tee sort.txt | grep r | tee match.txt | head`

15. Issue the **ls** command to view the contents of this redirectory.

    What did you notice?

16. View the contents of the **text files** that were created to see how the **tee** command
    was used in the previous pipeline command.

    What was the purpose of using the **tee** command for this pipeline command?

    You will now run a shell script to confirm that you properly issued that Linux pipeline command
    using the **tee** command and redirection.

17. Issue the following Linux command to run a checking script:
    `~uli101/week5-check-3`

    If you encounter errors, make corrections and **re-run** the checking script until you receive
    a congratulations message, then you can proceed.

18. Change to your **home** directory.

19. Remove the **~/redirect** directory and its contents.

    In the next investigation, you will learn various techniques to issue **multiple Linux commands**
    on the same line, or issue a **single Linux command over multiple lines**.

INVESTIGATION 3: ISSUING MULTIPLE UNIX/LINUX COMMANDS

In this investigation, you will learn how to issue multiple Unix / Linux commands in a single line or over multiple lines.

Perform the Following Steps:

1. Confirm you are located in your **home** directory in your Matrix account.

2. Issue the following Linux commands (using the *semicolon* character "**;**" to separate each Linux command):
   ```
   cal;pwd;date
   ```
   Note the output as well as the order of what each Linux command results.

3. Issue the following Linux commands: `(cal;pwd;date)`

   Was there any difference in the output of this command as opposed to the previous command?

   Let's see how grouping affects working with redirection.

4. Issue the following Linux commands: `cal;pwd;date > output.txt`

   What happened? Where is the output for the **date** command?
   Why isn't the output for the **cal** and **pwd** commands are NOT contained in that file?

5. Issue a Linux command to view the contents of the file called **output.txt**

   What do you notice?

   Let's use **grouping** to make modification to the previous command

6. Issue the following Linux commands: `(cal;pwd;date) > output.txt`

   What did you notice?

7. Issue a Linux command to view the contents of the file called **output.txt**

   What does *grouping* do when issuing multiple Linux commands (separated by a semi-colon ";") that uses redirection?

8. Issue the following Linux pipeline command (using \ at the end of most lines):
   ```
   echo "This will be split over multiple \
   lines. Note that the shell will realize \
   that a pipe requires another command, so \
   it will automatically go to the next line" |tr '[a-z]' '[A-Z]'
   ```
   Did the command work? What is the purpose of issuing a Linux command in this way?

9. Complete the Review Questions sections to get additional practice.

LINUX PRACTICE QUESTIONS

The purpose of this section is to obtain **extra practice** to help with **quizzes**, your **midterm**, and your **final exam**.

Here is a link to the MS Word Document of ALL of the questions displayed below but with extra room to answer on the document to simulate a quiz:

https://wiki.cdot.senecacollege.ca/uli101/files/uli101_week5_practice.docx

Your instructor may take-up these questions during class. It is up to the student to attend classes in order to obtain the answers to the following questions. Your instructor will NOT provide these answers in any other form (eg. e-mail, etc).

When answering Linux command questions, refer to the following Inverted Tree Diagram. The linux directory is contained in your home directory. Assume that you just logged into your Matrix account. Directories are underlined.

```
linux
|-- content
|   |-- assignments
|   `-- tests
|       |-- .answers.txt
|       `-- questions.txt
|-- projects
```

Review Questions:

1. Write a single Linux command to provide a detailed listing of all files in the **/etc** directory, sending the output to a file called listing.txt in the "**projects**" directory (append output to existing file and use a relative pathname)
2. Write a single Linux command to redirect the stderr from the command: **cat a.txt b.txt c.txt** to a file called **error.txt** contained in the "**assignments**" directory. (overwrite previous file's contents and use only relative pathnames)
3. Write a single Linux command: **cat ~/a.txt ~/b.txt ~/c.txt** and redirect stdout to a file called "good.txt" to the "tests" directory and stderr to a file called "**bad.txt**" to the "**tests**" directory. (overwrite previous contents for both files and use only relative-to-home pathnames).
4. Write a single Linux command to redirect the stdout from the command: **cat a.txt b.txt c.txt** to a file called wrong.txt contained in the "**projects**" directory and throw-out any standard error messages so they don't appear on the screen (append output to existing file and use only relative pathnames).

5. Write a single Linux **pipeline command** to display a detailed listing of the **projects** directory but pause one screen at a time to view and navigate through all of the directory contents. Use a relative-to-home pathname.
6. Write a single Linux **pipeline command** to display the sorted contents (in reverse alphabetical order) of the "**linux**" directory. Use a relative pathname.
7. Assume that the text file called "**.answers.txt**" contains 10 lines. Write a single Linux pipeline command to only displays lines 5 through 8 for this file. Use only relative pathnames.
8. Write a single Linux **pipeline command** to only display the contents of the "**assignments**" directory whose filenames match the pattern "**murray**" (both upper or lowercase). Use an absolute pathname.
9. Write a single Linux **pipeline command** to display the number of characters contained in the file called "**.answers.txt**". Use a relative-to-home pathname.
10. Write a single Linux **pipeline command** to display the number of lines contained in the file called "**questions.txt**". Use a relative pathname.
11. Write a single Linux **pipeline command** to display only the first 10 characters of each filename contained in your current directory. Also, there is will be a lot of output, so also pause at each screenful so you can navigate throughout the display contents. Use a relative pathname.
12. Create a **table** listing each Linux command, useful options that were mentioned in this tutorial for the following Linux commands: **cut** , **tr** , **wc** , and **tee**.

Author: Murray Saul

- This page was last edited on 16 June 2023, at 13:18.