OSL640: INTRODUCTION TO OPEN SOURCE SYSTEMS

WEEK 5: LESSON 1

ADDITIONAL LINUX COMMANDS
REDIRECTION SYMBOLS
/DEV/NULL FILE , THE HERE DOCUMENT

LESSON 1 TOPICS

Redirection – Part 1

- Additional Commands (**tr**, **cut**, **wc**)
- Concepts:
  - **Standard Input, Standard Output**, **Standard Error**
- Redirection Symbols: (**<**, **>**, **>>**, **2>**, **2>>**)
- Additional Redirection Concepts:
  - **/dev/null** File, The **Here Document**

Perform Week 5 Tutorial

- Investigation 1
- Review Questions (Questions 1 – 4)

ADDITIONAL FILE MANIPULATION COMMANDS

Additional Text File Manipulation Commands

Here are some additional commands to manipulate content of text files.

Refer to the table below for additional 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`

ADDITIONAL FILE MANIPULATION COMMANDS

Instructor Demonstration

Your instructor will now demonstrate using the following Linux commands:

- **tr**
- **cut**
- **wc**

REDIRECTION

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.*

For redirection, ***meta characters*** *are used.*

Redirection can be into a ***file*** *(shell meta characters are angle* ***brackets*** *'<', '>') or a* ***program*** *(shell meta characters are* ***pipe****symbol '|').*

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

REDIRECTION

Standard input (**stdin**) is a term which describes from where a command receives **input**.

The meta character "**<**" will redirect **stdin** into a command.

This would only apply to Unix/Linux commands that can **accept** *stdin* like **cat**, **more**, **less**, **sort**, **grep**, **uniq**, ***head***, ***tail***, **tr**, **cut**, and **wc**.

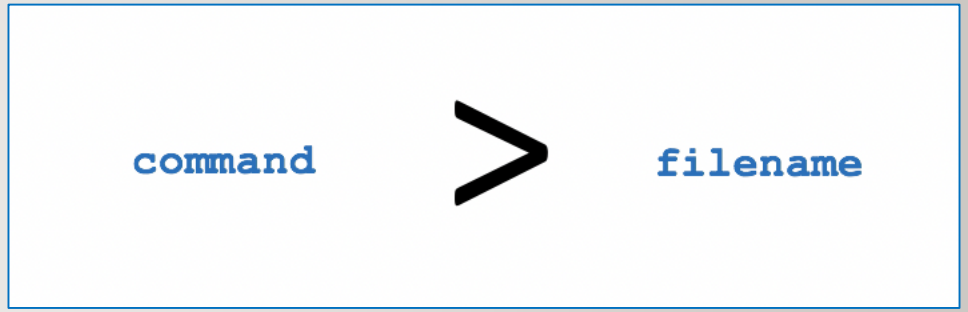
Examples:

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

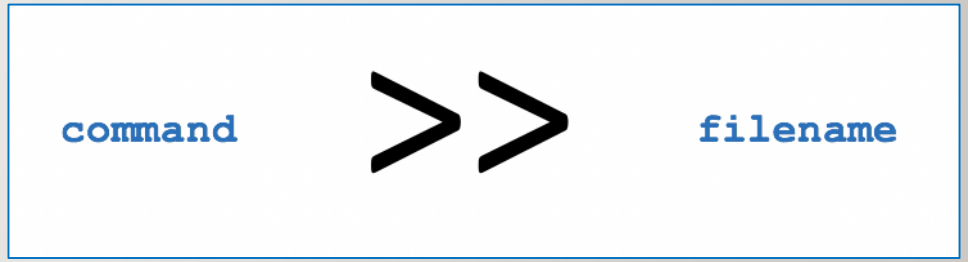
REDIRECTION

Standard output (**stdout**) describes where a command sends its **output**.

The meta character "**>**" will redirect **stdout** to a file either **creating** a new file if it doesn't exist or **overwriting** the content of an existing file.



The meta characters "**>>**" will redirect **stdout** to a file either **creating** a new file if it doesn't exist or **adding** stdout to the **bottom** to the existing file's contents.



Examples:

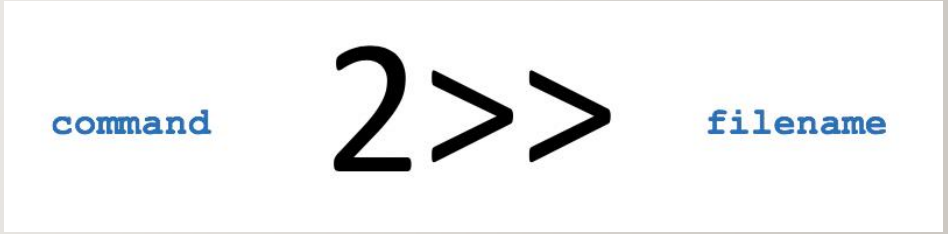
```
ls -l
ls -l > detailed-listing.txt
ls /bin >> output.txt
```

REDIRECTION

Standard Error (**stderr**) describes where a command sends its **error messages**.

The meta characters "**2>**" will redirect **stderr** to a file either **creating** a new file if it doesn't exist or **overwriting** the content of an existing file.

The meta characters "**2>>**" will redirect **stderr** to a file either **creating** a new file if it doesn't exist or **adding** stdout to the **bottom** to the existing file's contents.



Examples:

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

REDIRECTION

The **/dev/null** file (sometimes called the **bit bucket** or **black hole**) is a special system file that **discards** stdout or stderr.

This is useful to "*throw-away*" **unwanted** command output or errors.

Examples:

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

REDIRECTION

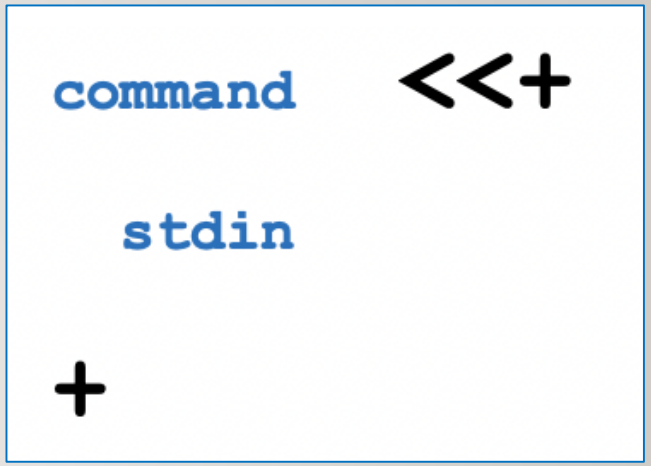
The **Here Document** allows stdin to be redirected into a command **within** the command-line.

The meta characters "**<<+**" will redirect **stdin** into the command.
The **+** symbol is used to identify the beginning and ending of the stdin.

You can use ANY symbol or series of characters to mark stdin as long as that symbols or characters are IDENTICAL and the ending symbol or characters are on a **separate** line with only that symbol or characters.

Example:

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



REDIRECTION

Instructor Demonstration

Your instructor will now demonstrate redirection:

- Standard Input
- Standard Output
- Standard Error
- Both Standard Output and Standard Error
- Both Standard Input and Standard Output
- Redirecting to **/dev/null**
- The **Here Document**

REDIRECTION

Getting Practice

To get practice perform **Week 5 Tutorial:**

- INVESTIGATION 1: BASICS OF REDIRECTION
- LINUX PRACTICE QUESTIONS (Questions 1 – 4)

ULI101: INTRODUCTION TO UNIX / LINUX AND THE INTERNET

WEEK 5: LESSON 2

PIPELINE COMMANDS
MULTIPLE / MULTILINE COMMANDS

LESSON 2 TOPICS

Redirection – Part 2

- Purpose of **Pipeline Commands**
- Linux Pipeline Command Syntax: |
- **tee** Command

Multiple / Multi-Line Commands

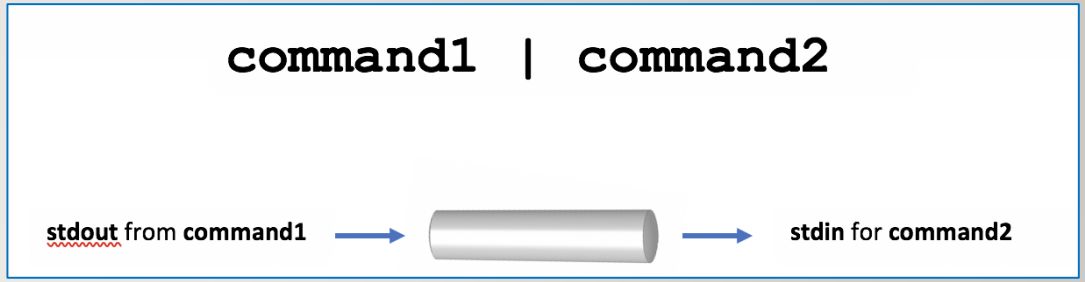
- Multiple Linux Commands using Semicolon ";" and Grouping: **( )**
- Issuing Large Linux Commands over Multiple Lines

Perform Week 5 Tutorial

- Investigations 2 and 3
- Review Questions (Questions 5 – 12)

PIPELINE COMMANDS

Pipeline Commands use a meta symbol "|" (called a pipe) to allow a command's **standard output** to be redirected into the **standard input** of other commands WITHOUT having to use **temporary** files.



Therefore, a few simple commands can be **combined** to form a more powerful pipeline command.

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
```

PIPELINE COMMANDS

Filters

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.

PIPELINE COMMANDS

Instructor Demonstration

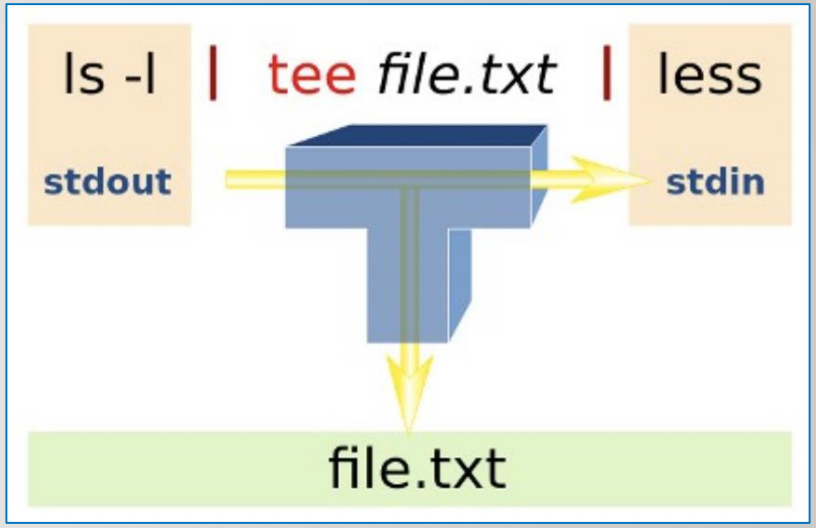
Your instructor will now demonstrate how to issue **Pipeline Commands**:

PIPELINE COMMANDS

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.



Examples:

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

PIPELINE COMMANDS

Instructor Demonstration

Your instructor will now demonstrate how to use the **tee** command within a **Pipeline Command**.

MULTIPLE / MULTI-LINE COMMANDS

There are ways that **multiple commands** can be run within a **single** command line.

```
command1 ; command2 ; command3
```

You can separate commands by using the **semi-colon character ";"**.

Example:

```
sleep 5; ls
```

Multiple commands can also be **grouped** by using **parentheses** to force commands to be run **together** (for example, to redirect **stdout** to a file)

Example:

```
(echo "Who is logged in:"; who) > whoson
```

(***Note:*** *all command output is sent to a file*)

MULTIPLE / MULTI-LINE COMMANDS

Commands may also be **spread-out over multiple lines**, making it easier (for humans) to interpret a long command.
You can add a **backslash** quoting symbol "\" at the end of a line.
The \ symbol "*quotes-out*" the meaning of the **ENTER** key as text (i.e. *new-line*) as instead of running the command.

```
command1 |   \
command2 |   \
command3
```

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]'
```

MULTIPLE / MULTI-LINE COMMANDS

Instructor Demonstration

Your instructor will now demonstrate how to issue Multiple Commands / Multi-Line Linux Commands:

- Multiple Linux Commands using semicolon ";"
- Multiple Linux Commands using Grouping ( )
- Mult-Line Linux Commands using Backslash \

PIPELINE COMMANDS
MULTIPLE / MULTI-LINE COMMANDS

Getting Practice

To get practice perform the online tutorial **Tutorial 5** (**ctrl-click** to open link):

- INVESTIGATION 2: REDIRECTION USING PIPES
- INVESTIGATION 3: ISSUING MULTIPLE UNIX/LINUX COMMANDS
- LINUX PRACTICE QUESTIONS (Questions 6 – 12)