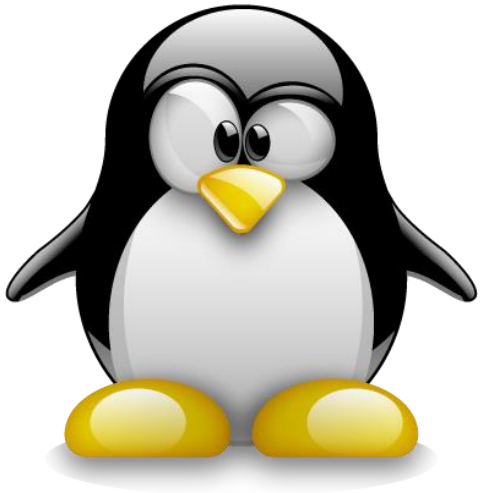
Linux Redirection and Pipes

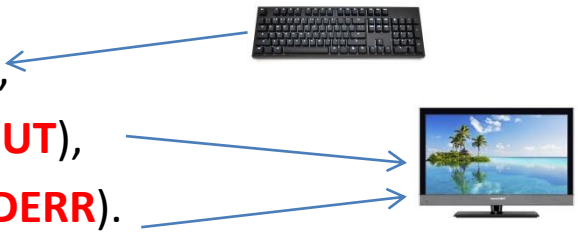
Outline

- STDIN (Standard Input)
- STDOUT (Standard Output)
- STDERR (Standard Error)
- Redirection
- Piping Commands



STDIN, STDOUT and STDERR

- File Descriptors
  - When the program is started,
    the shell gives all programs access to three “pointers” called
    - standard input (**STDIN**),
    - standard output (**STDOUT**),
    - and standard error (**STDERR**).



- Normally STDIN is connected to the keyboard,
  while STDOUT and STDERR are connected to the screen.
- However, with redirection (> , >>, <)
  you can change where a program gets its’ input from,
  and where it sends its output to.

STDIN, STDOUT and STDERR

- The redirection operator “>” lets you specify a file name to receive the STDOUT (output) of the command, while the “<“ operator lets you connect STDIN to a file containing the input.

Name	Abbreviation	File Descriptor	Standard Device
Standard Input	*stdin*	0	Keyboard
Standard Output	*stdout*	1	Console
Standard Error	*stderr*	2	Console

Examples of Redirection

`cat < file1.txt`	***Redirects from a file*** < tells cat to take the input from the file instead of the keyboard
`ls`	standards output for ls is the terminal
`ls > lsout.txt`	**Redirect Output of ls to a file**
`ls >> lsout.txt`	**Redirect and append output to a file**

Input and Output Redirection

- Default Standard Output is sent to the terminal.
- Redirect Standard Output to a file:
  - ls -al > martin.txt
  - ls -al 1> martin.txt [This is identical to the command above]
  - Please note that **>** and **1>** will overwrite anything in the martin.txt file if it exists, or create the file if it does not exist.
  - cat martin.txt
- Redirect standard input to a command.
  - cat < martin.txt
  - cat 0< martin.txt
- Count the number of lines in a file
  - wc -l martin.txt
  - wc -l < martin.txt

Error Redirection

- Default Standard Error is sent to the terminal
  - rm martin.txt
  - cat martin.txt
    If martin.txt does not exist we get an error message like…
    - cat: martin.txt: No such file or directory
- Redirecting Standard Error to a file:
  - cat martin.txt 2> errorfile.txt
  - cat errorfile.txt
    - cat: martin.txt: No such file or directory
- Redirecting both Standard Output and Standard Error to the same file:
  - cat ~/.profile martin.txt > logfile.txt 2>&1
    - [the .profile file exists; the martin.txt file does not exist]
  - cat ~/.profile martin.txt &> logfile.txt

Redirect Output or Error Stream to Append

- Redirect Standard Output to append (using >>)
  - ls -al > martin.txt
  - echo “Thank you” >> martin.txt

- Redirect Standard Error to append (using >>)
  - rm martin.txt
  - cat martin.txt
  - cat martin.txt 2> error.txt
  - echo “I don’t like errors” 2>> errors.txt

Practice

- echo “b” > file.txt
- echo “a” >> file.txt
- echo “d” >> file.txt
- echo “c” >> file.txt
- cat file.txt
- sort < file.txt
- **sort < file.txt > sortedfile.txt**
- cat softedfile.txt

- **In the bold command above, we have redirected both the standard input and standard output streams to a file**.

Redirection to standard Output or standard error

- Redirect to Standard Output
  - ls -al >&1
  - echo “Thank you” >&1 [this is the default behaviour anyway]
- Redirect to Standard Error
  - rm martin.txt
  - cat martin.txt >&2
  - echo “I don’t like errors” >&2
- Redirecting BOTH to standard output AND to standard error can be useful when piping commands

  [piping commands are coming up shortly]

Very useful redirection

- Discard the standard output of a command
  - Syntax: **command > /dev/null**
  - The special file /dev/null discards all data written to it
  - ls -al > /dev/null

- Redirect both to standard output AND to a file.
  - Syntax: **command | tee file**
  - ls -al | tee martin.txt

Piping Commands

- Send STDOUT of one process to STDIN of another process
  - Syntax: **command-1 | command-2 | … | command-N**
  - ls -al | grep file
  - ls –al /etc | grep x1
  - ls -al /etc | grep -i x1

- **grep**: print lines that match patterns
  - The **-i** option means ignore case sensitivity.
  - Very useful for searching contents of files and standard output.

Useful Piping Commands

- **grep**: print lines that match patterns
  - The **-i** option means match both upper and lower case
  - The **-c** option means display only the count of matching lines.
  - The **-n** option means show line number(s) and matching line(s)
  - The **-v** option means show all lines that do not match the search string.
  - For more options, look up the manual page (man grep)

- **sort**: sort the contents of a file alphabetically
  - The **-r** option means reverse sorting.
  - The **-n** option means sort numerically
  - The **-f** option means case insensitive sort.
  - The **-kN** option means sort by column N.

Piping Commands

- More examples:
- du -h | sort -h | tail -10

  - **du -h**: display disk usage for all folders recursively in the current folder *in human readable format (eg 1K, 3M, 5G)*
  - **sort -h**: sort lines of text in ascending order *comparing human readable numbers (eg 1K, 3M, 5G)*
  - **tail -10**: display the last part of a file or stream, *specifically the last 10 lines.*

  - Thus: this command displays the top 10 folders consuming the most disk space.

Piping Commands

- More examples:

- ls -al | grep ^d

  - Run the “ls –al” command and pipe the output in to grep
  - **grep ^d:** match all lines beginning with the character ‘d’

  - Thus: the command displays detailed information about all subfolders (and only subfolders) in the current working directory

Piping Commands

- More examples:

- ls -al | grep ^d -v | cut -c58-
  - Run the "**ls –al**" command and pipe the output in to grep.
  - **grep ^d -v**: the **minus v** option inverts the sense of matching, thus: match every line that does not begin with a 'd' character.
  - **cut -58-:** the cut command prints everything from each line starting at column 58 until the end of the line.

  - Thus: the command displays the names of the files (and only the files) in the current working directory.