ULI101

Week 05

Week Overview

- Simple filter commands: head, tail, cut, sort, tr, wc
- grep utility
- stdin, stdout, stderr
- Redirection and piping
- /dev/null file

head and tail commands

- These commands display the beginning or the end of a file respectively
- By default, 10 lines are displayed
  - The entire file will be displayed if it is less than 10 lines in length
- Example usage:

```
head [-line_count] file
```

for example: `head -3 users.log`

cut

- Selects which fields or columns to display from files or standard input
- Range can be specified in multiple ways:
  - 1–10 – first 10
  - 3–8 – 3rd to 8th
  - –10 – up to 10th
  - 2– – from 2nd until the end of line
  - 1–3,4,10– – combination of above
- Important options:
  - -c – cut **c**haracters

    Example: `cut -c1-2` – will cut first 2 characters

  - -f – cut **f**ields

    Example: `cut –f2,5` – will cut 2nd and 5th field

cut fields

- Default field delimiter is the tab
- Other field delimiter can be specified using the –d option

  For example:

  `cut-d,-f1-2` – will cut first 2 fields delimited with a comma
- Field delimiter must be a single character, only one character delimiters are supported
- If special characters are used for delimiters they must be quoted

  For example:

  `cut -d" " -f1` – space is the field delimiter

sort command

- Sorts files or standard input
- Is able to sort by fields
- Popular options:
  - -f – fold (ignore case in comparisons)
  - -n – numeric sort (default is ascii)
  - -u – display unique entries only
  - (do not display duplicate lines)
  - -r – reverse sort (default is lowest to highest value)

wc

- Counts the number of lines, words and/or characters in files or standard input
- Usage:

```
wc  option [filename]
```

- Options:
  - -l – count lines
  - -w – count words (delimited by whitespace)
  - -c – count bytes
  - -m – count characters
  - If no option is specified, line, word, and byte counts are displayed
  - Note than one extended ascii character is one byte

grep utility

- Searches for literal text and text patterns
  - Pattern-based searches will be covered in detail later in this course
- Example usage: `grep ford cars`
- Works with files or standard input
- Acts like a filter – outputs only lines which are successfully matched to a given regular expression
  - A successful match can be an entire line or any part of it, but the entire line will be displayed

Useful grep options

- -i – ignores case
- -n – numbers lines in the output
- -v – reverse match
- -c – displays count of matched lines

Standard Input and Standard Output

- Standard input (stdin) is a term which describes from where a command receives input
- Standard output (stdout) describes where a command sends it's output
- For most commands the default standard input and output are your terminal's keyboard and screen
- Standard input can be redirected from a file or piped from another command
- Most commands also accept a filename argument, which is internally redirected to standard input
- Standard output can be redirected to a file or piped to another command

Standard Input Redirection

```
command < filename
```

- Example:
  ```
  tr 'a-z' 'A-Z' < cars
  ```
- Used for commands which do not accept a filename as an argument

Standard Output Redirection

```
command > filename
```

- Redirects a command's standard output to a file
- Stdout redirection is represented by the **>** symbol

  Example:

  `ls > ls.txt` - will save output from the ls command into a file called ls.txt

- If the file exists already its content will be replaced
- To append (add) to a file, the **>>** symbol can be used

Standard Error

- In addition to standard input and standard output UNIX commands have standard error, where error messages are sent
- By default error messages are sent to the terminal
- Standard error can be redirected by using the 2> or 2>> redirection operators
- To redirect standard error to the same place as standard output, use 2>&1 redirection
- To redirect stdout to the same place as stderr, use >&2 redirection - this is how error messages are created in shell scripts

Inter-process communication

- Commands can send their standard output directly to standard input of other commands
- A few simple commands can be combined to form a more powerful command line
- No temporary files are necessary
- This is achieved by using pipes and tees

Pipes

- Pipes are represented by `|`
- Many commands can be "piped" together, filter commands are especially useful
  - Each filter processes the initial input based on it's design
  - Filters must be chained in a specific order, depending on what you wish to accomplish
- Example piping use:

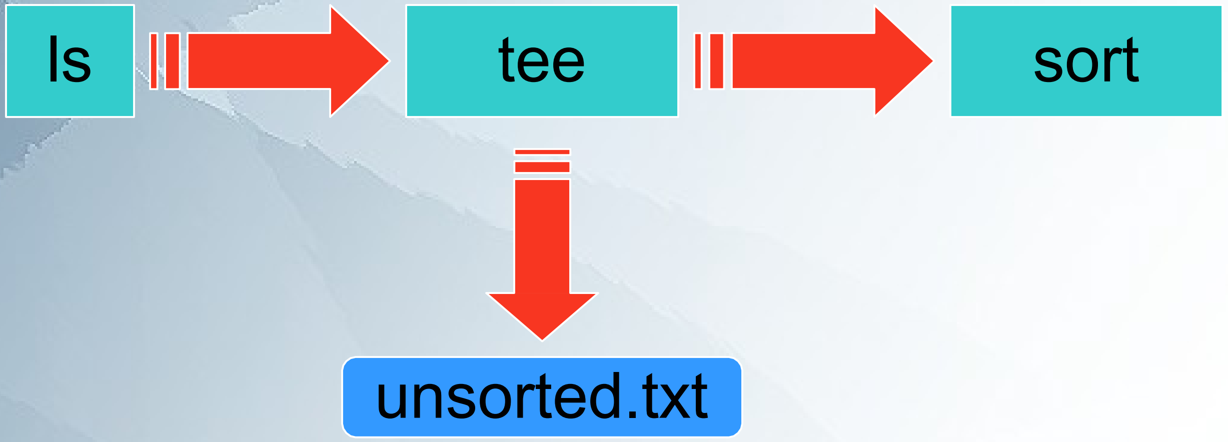
```
ls -al | more
```

Tee

- UNIX pipe with the tee utility can be used to split the flow of information

Example:

```
ls | tee unsorted.txt | sort
```



/dev/null file

- The /dev/null file (sometimes called the bit bucket or black hole) is a special system file that discards all data written into it
  - Useful to discard unwanted command output, for example:
    `find / -name "tempfile" 2> /dev/null`
- Also, /dev/null can provide null data (EOF only) to processes reading from it
  - Useful to purge (empty) files etc, for example: `cat /dev/null > ~/.bashrc`

"Here" documents

- The << symbol indicates a "here" document

Example:

```
sort << EOF
word
name
car
EOF
```

- Anything between EOF…EOF is sent to the standard input of a utility
- You can use some other string instead of "EOF"
- This is especially useful for embedding a small file within a shell script