



Western
UNIVERSITY • CANADA

IO and Redirection

Winter 2022

Streams

- Unix programs typically use three input and output streams by default:
 - STDIN – "Standard in". Data in this stream is read by the program. (0)
 - STDOUT – "Standard out". The program will write it's output to this stream (1)
 - STDERR – "Standard error". Warning and errors are written to this stream (2)

Redirection

- It is possible to use files instead of the default STDIN/STDOUT/STDERR
 - `<` - To use a file as input instead of the keyboard (STDIN). (The same as `0<`)
 - `>` - To use a file as output instead of the screen (STDOUT) (The same as `1>`)
 - `2>` - To use a file as error output instead of the screen (STDERR)

Redirection

- Examples
 - `cat file1.txt file2.txt > file3.txt`
Concatenate file1.txt and file2.txt and write the results to file3.txt.
 - If file3.txt exists, it is overwritten
 - If file3.txt does not exist, it is created
 - If there are any error messages, they are printed to the screen

Redirection

- Examples
 - `cat file1.txt file2.txt >> file3.txt`
Concatenate file1.txt and file2.txt and write the results to file3.txt.
 - If file3.txt exists, it is *appended*
 - If file3.txt does not exist, it is created
 - If there are any error messages, they are printed to the screen

Redirection

- Examples
 - `cat file1.txt file2.txt > file3.txt 2> file-error.txt`
Concatenate file1.txt and file2.txt and write the results to file3.txt.
 - If file3.txt exists, it is *overwritten*
 - If file3.txt does not exist, it is created
 - If there are any error messages, they are written to file-error.txt

Redirection

- Examples
 - Often, you want both the output and any errors to go to the same destination
 - Use the `2>&1` operator to tell the shell to redirect STDERR (2) to the same destination as STDOUT (1)

Redirection

- Examples
 - `cat file1.txt file2.txt > file3.txt 2>&1`
Concatenate file1.txt and file2.txt and write the results to file3.txt.
 - If file3.txt exists, it is *overwritten*
 - If file3.txt does not exist, it is created
 - If there are any error messages, they are written to file3.txt

Redirection

- Combined with the concept of background processes, this can be a powerful tool
- Suppose you had a long running process that generates some output. You could use redirection and job control to allow the process to run and to log any output to a file instead of the screen

Redirection

- Examples

- `cat file1.txt file2.txt > file3.txt 2>&1 &`
The same command as before, but the last `&` causes the process to run in the background allowing you to do other tasks -
- `tail -f file3.txt`
A common Unix idiom used to monitor the output of your background process as it runs. The `-f` means keep printing the tail of the file as it grows

Redirection

- Examples
 - `cat < file1.txt > file2.txt`
Read input from file1.txt instead of STDIN and write the output to file2.txt
 - (This is an odd example. cp would be the better tool. Using < is rarely used in practice since there are usually better methods)

Redirection

- There is a special file at `/dev/null` which you can use as an output file. It will "throw away" any input it receives. Essentially, this allows you to ignore any output generated
- There are also the special files `/dev/zero`, `/dev/random` which are sometimes used to generate input

Redirection

- Examples

- `cat file1.txt file2.txt > file3.txt 2>/dev/null`

The same command as before, but any warnings or errors will be redirected to /dev/null instead (that is, the output will be ignored)

- `cat file1.txt file2.txt > /dev/null 2>&1`


The same command as before but any output is ignored. This is a common idiom in Unix.

dustbin
↓

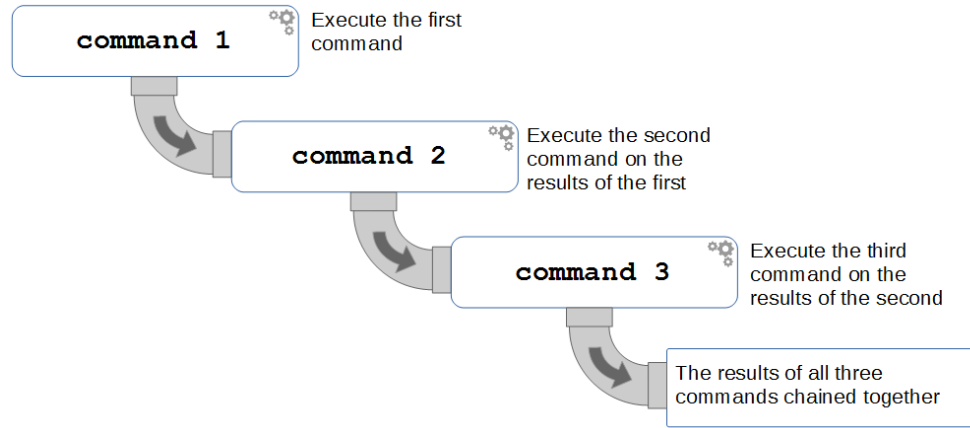
Pipes

- Remember, Unix can achieve complex tasks by "gluing" multiple tools together
- It could be possible to write the output from one command to a file, then run a second command that uses that file as input and write its output to a second file, and so on
- Pipes allow us to do this without worrying about redirecting to files at each step

Pipes

- Remember, Unix can achieve complex tasks by "gluing" multiple tools together
- We use the  symbol to denote a pipe
- E.g. `<command1> | <command2>`
 - This means the output from `<command1>` will be the input for `<command2>`
- (`|&` instead of just `|` can be used to include STDERR, but this is not common)

Pipes



Filters

- Many Unix commands are "filters". They read input, modify the data, and produce different output
- Usually they accept a file name as an argument, but if the file name is absent, they assume STDIN as input. This will be useful with pipes, but let's look at some common filters first

Filters

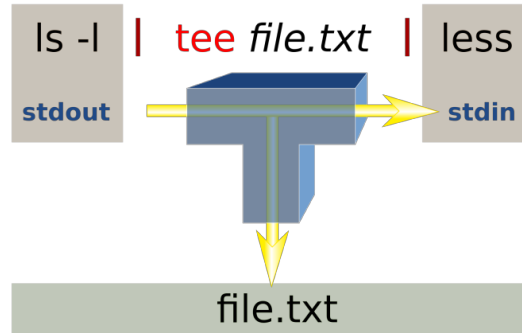
- Some common filters
 - `grep` – globally search for a regular expression and print matching lines
 - `tr` – translate one set of characters into another
 - `wc` – word count
 - `sort` – Sort the input

Filters

- Some common filters
 - cut – cut characters based on their position or based on a delimiter character ("remove columns")
 - paste – paste characters from an input file to each line from another input("add columns")
 - uniq – Remove duplicated lines ("unique")

Filters

- Some common filters
 - tee – Pass STDOUT back to STDIN but also write the output to a file (like a T-splitter)



Filters

- Some common filters
 - awk – A powerful programming language and tool for manipulating data
 - sed – stream editor. Another programming tool to manipulate data
 - (These tools are complex enough they could warrant their own course)

Filters

- Some common filters
 - We've used some filters already, just always with a file name supplied as an argument (e.g. cat, head/tail, less/more, strings)

Pipes with filters

- Filters are most powerful when used in a pipeline using pipes
- A simple example
 - `grep ^a /usr/share/dict/words | wc -l`
Find all the words in the Unix dictionary that begin with "a" and print the number of lines found to STDOUT

Pipes with filters

- A complex example

- ```
grep abc /usr/share/dict/words | sort -r |
tee file1.txt | cut -b2-6 | uniq
```

Find all words in the Unix dictionary that contain "abc",  
sort the matching words in <sup>-r</sup>reverse order, write the output  
tee file1.txt and STDIN, <sup>pipeline</sup>keep only characters 2-6, remove  
any duplicate lines, write the output to STDOUT

*uniq*

*pipeline.*



Western  
UNIVERSITY • CANADA