MODULE 3 :
Standard I/O, Redirection Pipes and Filters

Standard I/O in Unix

- **standard input (0: stdin)**
  - ✓ The default place where a process reads its input (usually the terminal keyboard)
- **standard output (1: stdout)**
  - ✓ The default place where a process writes its output (typically the terminal display)
- **standard error (2: stderr)**
  - ✓ The default place where a process can send its error messages (typically the terminal display)

Stream Redirection

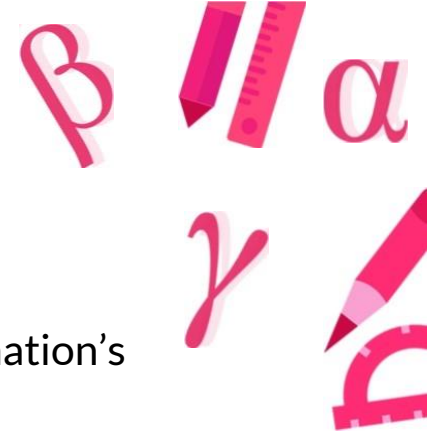
Unix includes redirection commands for each stream. These commands write standard output to a file.

Commands with a single bracket overwrite the destination's existing contents.

Overwrite

- `>` - standard output
- `<` - standard input
- `2>` - standard error

Stream Redirection(contd.)



Commands with a double bracket do not overwrite the destination's existing contents.

Append

- `>>` - standard output
- `<<` - standard input
- `2>>` - standard error

If a non-existent file is targeted (either by a single-bracket or double-bracket command), a new file with that name will be created prior to writing.

Redirecting Standard I/O

- Standard input and output can be redirected
  - Lets you combine programs and Unix tools
- Standard input is redirected from a file using `<`

```
wc < standard_input.txt
```

Any use of stdin will instead use standard_input.txt in this example

Redirecting Standard I/O (contd.)

standard_input.txt

```
f af asfsdf afdsf
sdf gvddfg ssdffg
 sdgf asff assdf
```

$ wc < standard_input.txt

```
 3  10  54
```

Output : Lines Words Characters

Redirecting stdout

- Standard output is redirected to a file using >

```
cal 2007 > calendar.2007
```

```
wc -l < standard_input.txt > Lines.out
```

- the ***stdout*** of wc -l is directed to file `Lines.out`

```
$ wc -l < standard_input.txt > Lines.out

$ cat Lines.out

3
```

Redirecting stderr

Standard error can be redirected with 2>

```
root in /home/akhan/Documents/demo/important
_ ls file1.txt file2.txt
file1.txt  file2.txt

root in /home/akhan/Documents/demo/important
_ ls File.*
ls: cannot access 'File.*': No such file or directory

root in /home/akhan/Documents/demo/important
_ ls File.* > output.txt
ls: cannot access 'File.*': No such file or directory

root in /home/akhan/Documents/demo/important
_ cat output.txt

root in /home/akhan/Documents/demo/important
_ ls File.* 2> output.txt

root in /home/akhan/Documents/demo/important
_ cat output.txt
ls: cannot access 'File.*': No such file or directory
```

Appending to a File

- The `>>` operator appends to a file rather than redirecting the output to a file

```
root in /home/akhan/Documents/demo/important
_ cat -n file1.txt
     1  hello from file1

root in /home/akhan/Documents/demo/important
_ echo "line2 hello from file1" >> file1.txt

root in /home/akhan/Documents/demo/important
_ cat -n file1.txt
     1  hello from file1
     2  line2 hello from file1
```

Pipes

Pipes allow ***stdout*** of one program to be ***stdin*** of another

- The output of the command to the left of '|' becomes the input of the command on its right

```
root in /home/akhan/Documents/demo/important/demo
_ ls -l | sort
-rw-r--r-- 1 root root 0 Oct 18 17:40 file1
-rw-r--r-- 1 root root 0 Oct 18 17:40 file2
-rw-r--r-- 1 root root 0 Oct 18 17:40 file3
-rw-r--r-- 1 root root 0 Oct 18 17:40 file4
-rw-r--r-- 1 root root 0 Oct 18 17:40 file5
-rw-r--r-- 1 root root 0 Oct 18 17:40 file6
-rw-r--r-- 1 root root 0 Oct 18 17:40 file7
-rw-r--r-- 1 root root 0 Oct 18 17:40 file8
-rw-r--r-- 1 root root 0 Oct 18 17:40 file9
total 0

root in /home/akhan/Documents/demo/important/demo
_ ls -l | sort -r
total 0
-rw-r--r-- 1 root root 0 Oct 18 17:40 file9
-rw-r--r-- 1 root root 0 Oct 18 17:40 file8
-rw-r--r-- 1 root root 0 Oct 18 17:40 file7
-rw-r--r-- 1 root root 0 Oct 18 17:40 file6
-rw-r--r-- 1 root root 0 Oct 18 17:40 file5
-rw-r--r-- 1 root root 0 Oct 18 17:40 file4
-rw-r--r-- 1 root root 0 Oct 18 17:40 file3
-rw-r--r-- 1 root root 0 Oct 18 17:40 file2
-rw-r--r-- 1 root root 0 Oct 18 17:40 file1
```

Regular Expressions and Metacharacters

Regular expressions are a powerful means for pattern matching and string parsing that can be applied in so many instances. With regular expressions you can:

- Validate text input
- Search (and replace) text within a file
- Batch rename files
- Undertake incredibly powerful searches for files
- Interact with servers like Apache
- Test for patterns within strings
- And much more

What makes up Regular Expressions ?

There are two types of characters to be found in regular expressions:

- Literal characters
- Metacharacters

Literal characters are standard characters that make up your strings. Every character in this sentence is a literal character. You could use a regular expression to search for each literal character in that string.

Metacharacters are a different beast altogether; they are what give regular expressions their power. With metacharacters, you can do much more than searching for a single character. Metacharacters allow you to search for combinations of strings and much more.

What makes up Regular Expressions ? (contd.)

- ^ Indicates the beginning of an input string
- $ Indicates the end of the an input string
- * Indicates the preceding subexpression is to be matched zero or more times
- + Indicates the preceding subexpression is to be matched one or more times
- ? Indicates the preceding subexpression is to be matched zero or one time
- {n} Match exactly n times (Where n is a non-negative integer)
- {n,} Match at least n times (Where n is a non-negative integer

What makes up Regular Expressions ? (contd.)

- {n,m} Match at least n and at most m times (Where m and n are non-negative integers and n <= m)
- . Matches any single character except “n”
- [xyz] Match any one of the enclosed characters
- x|y Match either x or y
- [^xyz] Match any character not enclosed
- [a-z] Matches any character in the specified range.
- [^a-z] Matches any character not in the specified range

Mixing Inputs from Standard Input and a File

When we type something into our terminal program, we'll often see output. For example:

```
$ echo hello
hello
```

As we can see above, `echo hello` is the command that output "hello", but where has the output really go?

The ***default*** place for the output to go is the shell screen which you are connected to. It is the space with which we are informed about the responses of the commands, even if they weren't able to execute successfully.

Mixing Inputs from Standard Input and a File (Contd.)

Consider the example of wc command.

- We are counting the number of lines in file1.

- Then we are streaming the contents of file1 to the same wc command, with standard input redirection (**stdin**)

- Next we are streaming the **stdout** of the command to a new file file1.line_count.

Mixing Inputs from Standard Input and a File (Contd.)

```
root in /home/akhan/Documents/demo/important
_ whatis wc
wc (1)               - print newline, word, and byte counts for each file

root in /home/akhan/Documents/demo/important
_ cat file1.txt
line1 hello from file1
line2 hello from file1
line3 hello from file1
line4 hello from file1
line5 hello from file1

root in /home/akhan/Documents/demo/important
_ wc -l file1.txt
5 file1.txt

root in /home/akhan/Documents/demo/important
_ wc -l < file1.txt
5

root in /home/akhan/Documents/demo/important
_ wc -l < file1.txt > file1.line_count

root in /home/akhan/Documents/demo/important
_ cat file1.line_count
5
```

Filters

Filters are programs that take plain text(either stored in a file or produced by another program) as standard input, transforms it into a meaningful format, and then returns it as standard output.

Linux has a number of filters. Some of the most commonly used filters are:

- `cat` **:** Displays the text of the file line by line.

- `head` **:** Displays the first n lines of the specified text files. If the number of lines is not specified then by default prints first 10 lines.

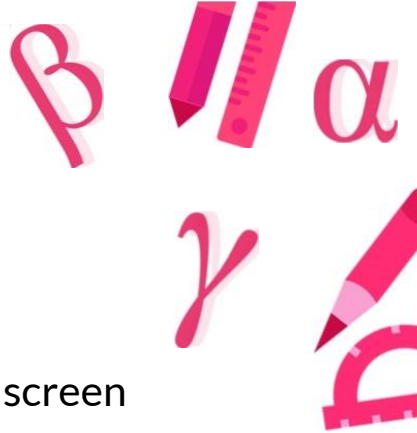
Filters (Contd.)

- tail : It works the same way as head, just in reverse order. The only difference in tail is, it returns the lines from bottom to up.
- sort : Sorts the lines alphabetically by default but there are many options available to modify the sorting mechanism.
- uniq : Removes duplicate lines.
- wc : wc command gives the number of lines, words and characters in the data.

Filters (Contd.)

- grep : grep is used to search a particular information from a text file.
- sed : sed stands for stream editor. It allows us to apply search and replace operation on our data effectively. *sed* is quite an advanced filter.
- tr : tr stands for translate, it is used for translating or deleting characters

tee Command



`tee [-a] file`

The tee command can be used to send standard output to the screen and to a file simultaneously.

`make | tee build.log`

Runs the make command and stores its output to build.log.

`make install | tee –a build.log`

Runs the make install command and appends its output to build.log

Terminal (/dev/tty) and Trash (/dev/null) Files

- Both `/dev/tty` and `/dev/null` are character device file.
- tty is a special file representing the '*controlling terminal*' for the current process.
- `/dev/tty` is a special file, representing the terminal for the current process. So, when you `echo 1 > /dev/tty` -> your message (1) will appear on your screen

Terminal (/dev/tty) and Trash (/dev/null) Files (contd.)

`/dev/null` is a special file called the null device in Unix systems. Colloquially it is also called the '***blackhole***' because it immediately discards anything written to it and only returns an end-of-file.

`echo 1 > /dev/null` -> your message ('1') will disappear.

Revisiting uniq, sort & tr commands

1. uniq : Removes duplicate lines.

2. sort : Sorts the lines alphabetically by default but there are many options available to modify the sorting mechanism.

3. tr : tr stands for translate, it is used for translating or deleting characters. It supports a range of transformations including uppercase to lowercase, squeezing repeating characters, deleting specific characters and basic find and replace.

Thank You!