UNIX and C

1 The Shell

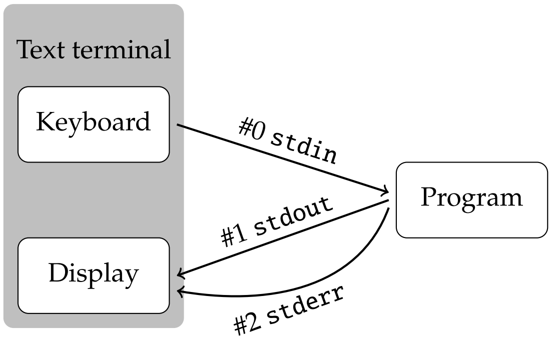
- A powerful way to perform work on a computer through a text interface
  - Run programs
  - Control how the programs work
- Perform sequences of commands to achieve even more complex work
- There are many choices for which shell to use, the most popular of which is `bash` (bourne again shell)

2 Ethos of the UNIX Shell

- Not one monolithic program
- Instead many small programs
- Allow user to combine these together to make new functionality
  - Using pipes
  - Using script files

3 `stdin`, `stdout` and `stderr`

- Remove the need to worry about I/O devices
- Two types of output, each can be redirected
- These are stream variables, can redirect e.g. `2>&1`



4 Pipes

- The shell provides you with many small ‘tools’
  - The power comes from composing them together
  - Pipes provide a means to do this
  - Each command takes input (from the keyboard)
  - Each command produces output (to the screen)
- We can redirect input or output
  - < take input from a file
  - > write output to a file
  - `|` take the output of one command and use as input to next

5 Output pipes

- Add ">" or ">>" and the name of a file after your command before you hit "Enter/Return" – e.g. "`ps > file.txt`"
- If the file doesn't exist already, it will be created for you in the directory in which you are working
- ">>" appends, ">" overwrites – so be careful when using ">"!!

6 `grep`

- `grep` is a search tool that uses regular expressions for matching
  - `grep "help" file.txt`
    * Lists all lines in `file.txt` containing the word '`help`'

7 Regular Expressions

- A concise way to match different strings
  - * - match any number of the proceeding character
  - ? - match one of the proceeding character
  - + - match one or more of the proceeding character
  - `[ABC]` - class as single character
  - `[A-Z]` - all the upper case characters `A` to `Z`
- e.g. `[A-Za-z]*[0-9].txt`

8 `sort`

- Sorts a file (if specified)
  - `stdin`: standard input, by default from terminal
- Where does it put the results?
  - `stdout`: standard output, by default the terminal
  - or a file with `-o filename`

9 `tr` - translate

- `tr SET1 SET2`
  - translates or deletes characters from `SET1` to `SET2`
  - e.g. `tr 'A-Z' 'a-z'` makes a lower case version of `stdin`
  - option `-c` takes complement of `SET1`
  - option `-s` squeezes repeats to a single character
  - option `-d` deletes all characters in `SET1`
  - e.g. `tr -dc '[:print:]'` - deletes all non printable characters

10 Options and more options. . .

- Most UNIX commands have many options
- To find out what these are:
  - Ask the command
    * e.g. `sort --help`, `grep -h`
  - Refer to the manual
    * e.g. `man sort`, `man tr`
  - Go online
    * e.g. Search command in Google

11 uniq

- Remove or report repeated lines
- Use with sort to find lines repeated throughout document
- e.g. sort | uniq
- Use -c option to count number of repetitions
- Tie these all together: what does this do?

-
  ```
  tr 'A-Z' 'a-z' < infile | tr -cs 'a-z' '\n' | sort \
        | uniq -c | sort -n!
  ```

12 Defining our own UNIX command

- UNIX commands are just executables, most of which are written in C
- Suppose we want to count only frequent words, we could write a filter function to forward lines starting with a number above a certain value

```
#include <stdio.h>

int main(int argc, char* argv[]){

  int limit;
  sscanf(argv[1], "%d", &limit);

  char* line = NULL;
  size_t size=0;
  while(getline(&line, &size, stdin) >0){
    int number = 0;
    sscanf(line,"%d", &number);
    if(number>= limit){
      printf("%s", line);
    }
  }
return 0;
}
```

13 Formatting (stdio.h)

printf(char *format, ...)	To stdout
fprintf(FILE *stream, char *format, ...)	To file/stream specified
sprintf(char *str, char *format, ...)	• Write into string/array specified • String needs memory to be allocated already
scanf(char *format,...)	From stdin to specified variables
fscanf(FILE *stream, char *format,...)	• From specified file • scanf(...) is the same as fscanf(stdin,...)
sscanf(char *str,char *format,...)	From a given string

14 getline()

- From GNU C lib and is more reliable than gets()
- Parameters
  - Pointer to malloc()'d block for result
    - will malloc() if NULL
  - Pointer for number of bytes in the malloc()'d block
  - Stream to read from

15 Using our program in UNIX

- Compile: `gcc filter.c -o filter`
- Put in a pipe e.g.

```
tr 'A-Z' 'a-z' < filter.c | tr -cs 'a-z' '\n' \
       | sort | uniq -c | sort -n | ./filter 3

wc * | sort -n | ./filter 10
```

16 Be robust

- Check the number of command line parameters
- Report problems to stderr
- Return value of main()
- For more complex joining of UNIX commands use shell script

17 File handling

- Files are stored in a hierarchical structure
- Allows grouping
- Navigation
  - ls - list the current folder
  - cd - change folder
  - mkdir - make new folder
  - mv - move a file / folder
  - cp - copy a file / folder
  - rm - delete a file
  - rmdir - delete a folder
  - du - how much space does a folder / file take?
  - find - list all files

18 Shell scripts

- A Shell Script is simply a collection of commands enclosed in a file
- Useful for when you have to type lots of commands to do one thing
- Whilst this is not impossible, it can get rather time-consuming
- Putting all the commands into a shell script enables them to be executed at the command line in one single command

19 Writing a Shell Script

- You can write shell scripts in any text editor of your choosing
- They should be saved with a `.sh` extension, e.g. `myscript.sh`
- They must all begin with the line `#!/bin/bash`
  - "`#!`" tells UNIX this is a script that can be run
  - `/bin/bash` tells Linux what program to run the script with

20 Example

- This script creates a new directory, changes into it and creates two new text files

```
#!/bin/bash
mkdir newDirectory
cd newDirectory
touch file1.txt
touch file2.txt
```

21 How do you run a shell script?

- Firstly, you need to make sure you have permission to execute the script file Use the `chmod` command to do this
  - `chmod a+x myscript.sh`
- Then, at the command line, type `./scriptname` and your script should run
  - e.g. `./myscript.sh`

22 Doing things to multiple files

- A handy little tool for doing the same operation to lots of files

```
#!/bin/bash
for f in *
do
  #something in here
  echo $f
done
```

23 Parameters

- You can add parameters to a script when you run them
- `./myscript.sh foo bar`
  - "`foo`" and "`bar`" are the parameters here
- Refer to them using the `$` sign in scripts
  - `$1`, `$2`, etc.

24 The `if` statement in shell scripts

```
#!/bin/bash
if [ $1 -lt $2 ]
then
  echo "yes" $1 "is less than" $2
else
  echo "no it isn't"
fi
```

- The `else` bit is optional
- Uses `==`, `!=`, `-gt`, `-lt`, `-le`, `-ge` for equality, inequality, greater than, less than, less than or equal, greater than or equal

25 Some last bits

- `if [ -a FILE ]` - true if `FILE` exists
- `if [ -z STRING ]` - true if `STRING` is empty
- Variables:
  - `VAR="Hello World"`
  - `echo $VAR`
  - ``TD="The time is `date`"``
  - `echo $TD`
    - `The time is Wed 20 Nov 15:44:14 GMT 2019`