Intro to bash scripting

- Here we’re talking about storing bash code in a file (script) to be run when desired – you can also use this bash syntax typed directly on command line too
- At start of bash scripts, need to include a line specifying which interpretter is supposed to be used to read script, e.g. #! /bin/bash
- Other than the hash bang (#!) above, comments in a bash script start with # then go to the end of line (like C++ //)

Basic syntax

- More or less, any command you can type at the keyboard can be put on a line of a bash script, and vice versa, when that line in the script is reached, that command runs, e.g.

```
#! /bin/bash
g++ foo.cpp -o foo # compiles file foo.cpp in curr dir
ls -l foo* # lists files (in curr dir) beginning with foo
```

- In some ways bash is very flexible with whitespace, in others it is very restrictive (details as they're relevant)
- Note that brackets and commas not used when passing arguments (same later when we get to calling functions)

Variables

- Global by default, have alphanumeric names (start with an alpha), and automatically declared when first used, e.g. myvar=3
- Variables are all of type text string, since based on idea of typed user input, but certain arithmetic operations permitted
- Picky whitespace: you cannot have space on either side of the =
- Variable names act somewhat like a reference, to use the content stored in a variable we use $ to deref, e.g.

```
y=3 # assigns 3 to y
x=$y # lookup value of y and assign to x
```

Output with printf or echo

- To output text can simply use echo command, automatically prints newline at end:

```
echo “value of variable x is $x”
```

- Alternatively, can use printf and \n’s, similar to C

```
printf “value of x is %x\n”, x
```

Input with read

- Can read line of user input into variables using read

```
read x y z
```

- Note that reads first word into x, second into y, and the entire rest of the line into z
- Can use -p option to display a prompt then read, e.g.

```
read -p “enter some text” x
```

- Various other options also available

Command redirection

- All the command redirection we’ve discussed earlier still works in bash scripts,
- e.g. to run program p, taking input from file1 and sending output to file2:

```
p < file1 > file2
```

- e.g. to pipe p’s output into q, then q’s into filex:

```
p | q > filex
```

Here strings <<

- You can run a command and tell it to read its input from a string using the here-string <<<, e.g.

```
mycommand <<< “blah blah blah”
```

- You can give it a string that spans multiple lines of input by specifying a string to mark the end of the input, say LASTWORD, then using <<LASTWORD to start, e.g.

```
mycommand <<LASTWORD
blah blah blah
and more blah blah blah
LASTWORD
```