References, quotes, and brackets

- Bash variable names act as references, with $ to dereference or evaluate, e.g.

```
A=1 # assign 1 to A
B=$A # copy 1 into B
C=A # C is now also a reference to A
A=2 # change A's value to 2
echo "$A, $B, $C, $(($C))" # prints 2, 1, A, 2
```

- Note that the $ can be applied to expressions in brackets to evaluate them, e.g. $(command), $((mathexpression))

Single and double quotes

- Double quotes (weak quotes) allow interpretation of a string contents, e.g. in “the value of x is $x” the $x gets replaced with the current value of x
- Single quotes (strong quotes) prevent interpretation of characters, e.g. in ‘the value of x is $x’ the $x does not get intepretted, it really is the character $ followed by the character x
- The \’ inside single quotes DOES get interpretted, so we are able to embed a single quote inside a single-quoted string

Square brackets in bash

- Square brackets in bash (and linux) can actually be used as a stand-alone test (true/false) command, e.g.

  [“foo” = “foo”]

- Most commonly seen as part of if statements, loops
- In fact, [is the (oddly named) command, the rest are args
- Old bash syntax also uses $[expr] for delimiting expression evaluation, e.g. echo “$[$x]”, modern syntax is $((expr))

Double square brackets

- A generally more flexible conditional syntax is given by double square brackets, e.g. [[expr]]
- We'll see these later when we do compound boolean expressions, regular expression comparisons, etc

Single round brackets

- $( ) can be used to run a command in a subshell of its own and capture the output, e.g. x=$(foo)
- This is handy, in that if foo crashes then it only crashes the subshell, it doesn’t also crash our current script
- Anything that foo tried to print to stdout actually goes into x
- If we want to capture the exit/return value of foo afterwards, we’ll find it in the special variable $?
- Single round brackets will also be used for arrays, discussed later

Double round brackets

- Double round brackets are used to enclose integer math operations in bash, and preceding with the $ sign lets use capture the result, e.g.

```
a=3
b=4
a=$((41 / a + b--))
echo “$a” # displays 17
```