Regular expressions

- Bash also provides a means of comparing strings against patterns, with the =~ operator to do the comparison and some slightly different syntax for describing the patterns
- This is often used when checking text (e.g. parameters) to see if they have a valid format before processing them
- Generally we’ll specify our patterns inside single quotes

Basic sequences and matching

- The simplest form of pattern is just a specific string, e.g. ‘blah’, which would match any string CONTAINING blah
- We use the =~ (inside [[ ]]) to test for a match, e.g.

```
Function containsblah() {
    local param=$1
    local pattern=’blah’
    if [[ $param =~ $pattern ]] ; then
    echo “$param contains $pattern”
    fi
}
```

Specifying a set of characters, [ ]

- We can use syntax like [xyz] to specify the character we want can be any of the ones inside the square brackets, x, y, or z in this case.
- We can also specify ranges, e.g. [a..z] matches any character from a to z
- The ^ can be used to invert this, specifying anything except the characters listed, e.g. [^1..9] means anything except the digits 1 through 9

Repeating patterns

- We can specify that a pattern can repeat a certain number (or range) of times
- (pattern)* specifies it can repeat 0 or more times
- (pattern)? specifies it can repeat 0 or 1 times
- (pattern){m,n} specifies it can repeat m to n times
- (pattern)+ specifies it can repeat 1 or more times

Matching the ends of a string

- Sometimes we want to specify a pattern must come at the start of the string, this is done using ^pattern
- Sometimes we want to specify a pattern must come at the end of the string, which is done using pattern$
- If we don’t include the ^ and/or $ then the pattern will match any string containing the pattern, which may have undesirable extra characters on either side of the pattern

OR with patterns (|)

- Sometimes we want to specify the next part of the string could look like either one of two patterns, this can be done using pattern1 | pattern2

Example: specifying a positive int

- Suppose we want a string that represents a positive integer, with no leading 0’s
- The first character would be a 1..9, then there could be 0 or more characters that were each a 0..9
- There can’t be anything before or after the integer part, so we need to use the ^ and $ around our pattern
- A valid pattern string would thus be ‘^[1..9][0-9]*$’

Example: specifying a time

- Suppose we want to specify that a string to represent a time in the form hh:mm, in 24-hour format (say 00:00 through 23:59)
- If the first digit is a 0 or 1, the second digit can be 0-9, but if the first digit is a 2 then the second digit can only be 0-3
- The third digit can be 0-5, the final digit can be 0-9
- ‘^(([01][0-3])|([2][0-9]))[:][0-5][0-9]$’