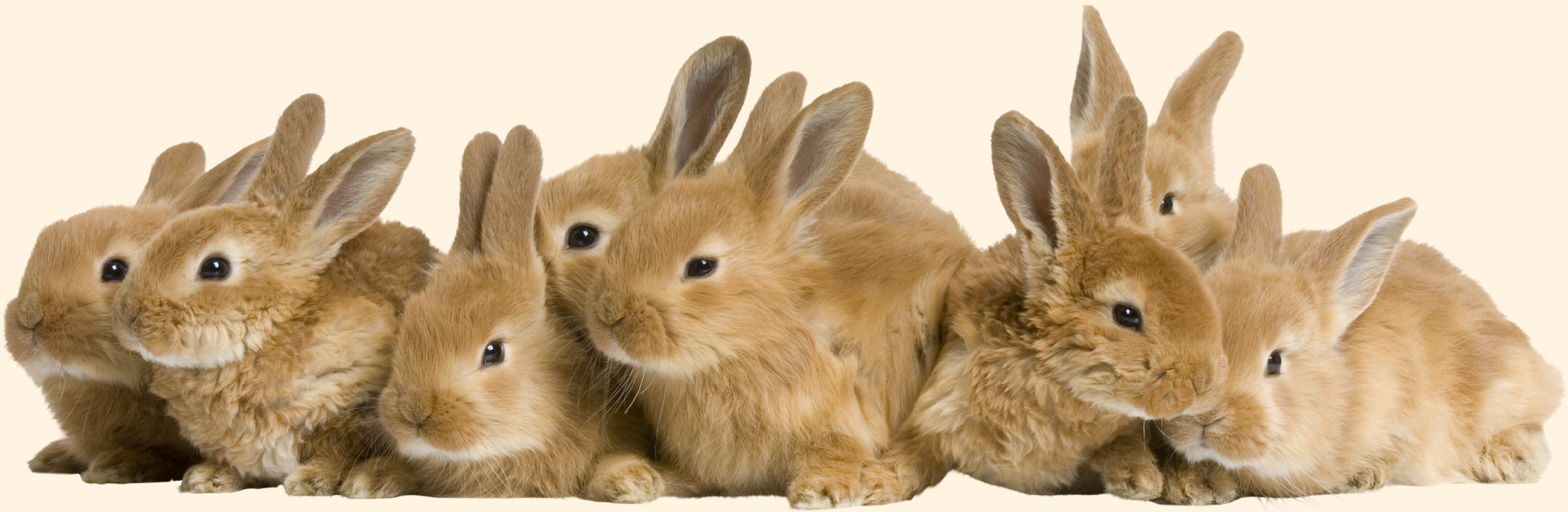
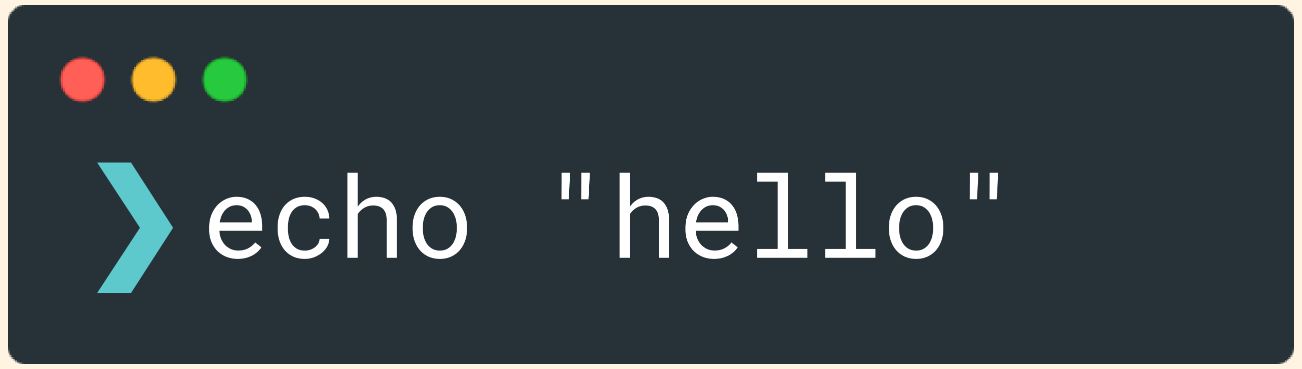
Expansion & Substitution



echo

The echo command is very simple. It simply displays text that we pass to it. We'll be using it to demonstrate some concepts in this section.

It's particularly useful in shell scripts (we cover them later), when we want to output something to the screen from within a file.



```
❯ echo "hello"
```

Wildcard Characters (aka globbing patterns)

We can use special wildcard characters to build patterns that can match multiple filenames at once.

The asterisk (*) character represents zero or more characters in a filename. For example...

- ls *.html will match any files that end with .html like index.html and navbar.html

- cat blue* will match any files that start with "blue" like "blue.html" or "bluesteel.js"

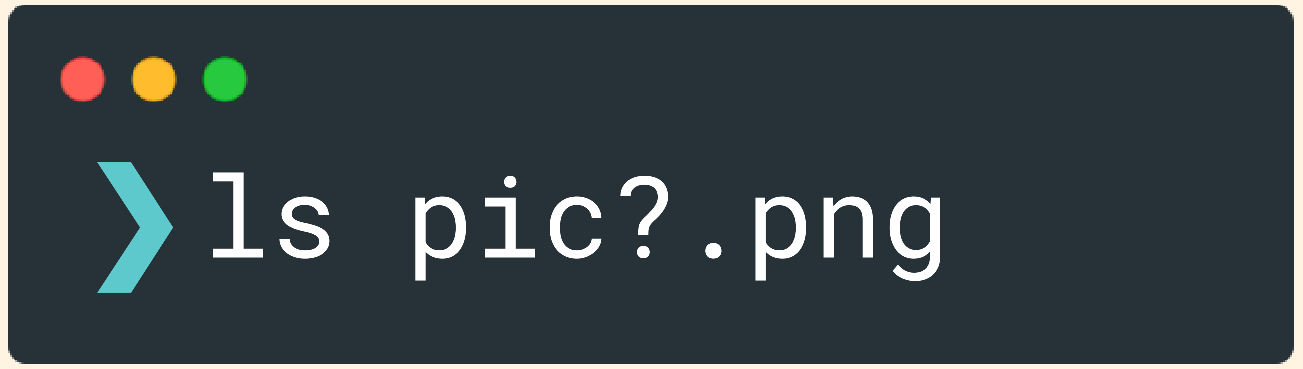
```
❯ls *.html
```

The ? Wildcard

The question mark (?) character represents any single character.

- ls app.?? will match any files named "app" that end with two character file extensions like "app.js" or "app.py" but NOT "app.css"

- ls pic?.png will match pic1.png, pic2.png, pic3.png, but also picA.png, or picx.png.



```
❯ ls pic?.png
```

Range Wildcards

Inside of square brackets ([ ]) we can specify a range of characters to match.

- ls pic[123].png will only match pic1.png, pic2.png, and pic3.png

- ls file[0-9] will match file1, file2, file3, up to file9

- ls [A-F]* will match any files that begin with a capital A, B, C, D, E, or F

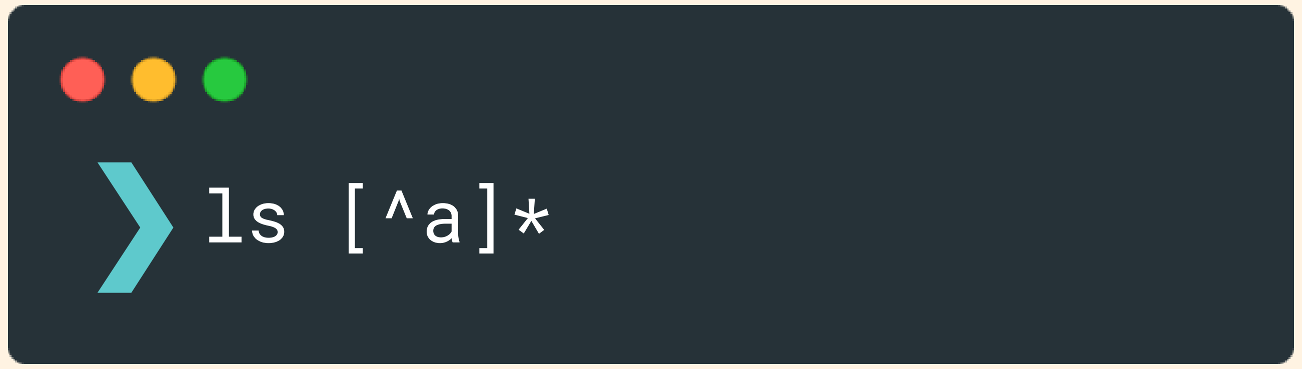
```
❯ls [A-F]*
```

Negating Ranges

Inside of square brackets ([ ]) we can specify a range of characters to NOT match, using a caret (^)

- ls [^a]* will match any files that do NOT start with "a"

- ls [^0-9]* will match any files that do NOT start with a numeric digit (0-9)

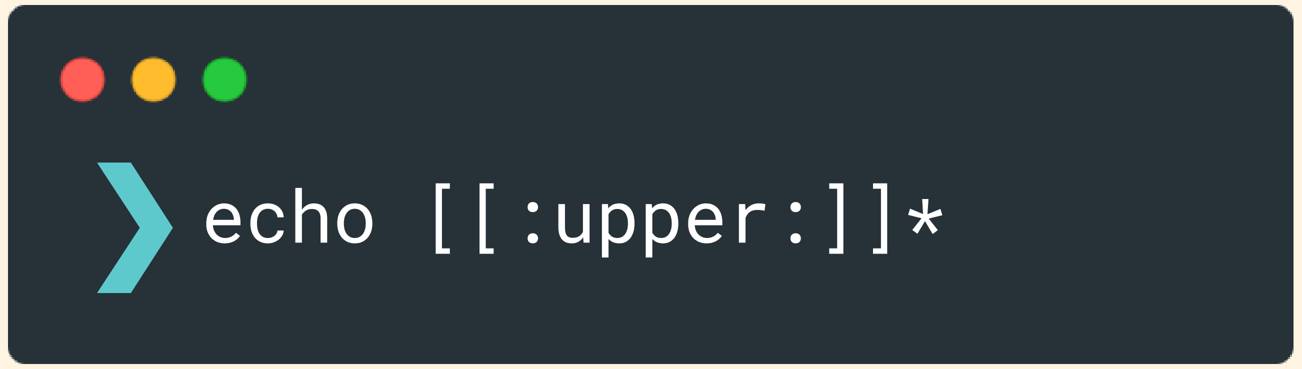


```
❯ ls [^a]*
```

Character Classes

We can also use predefined named characters classes:

[:alpha:] - alphabetic characters, upper and lower
[:digit:] - digits 0-9
[:lower:] - lower case letters
[:upper:] - upper case letters
[:blank:] - blank characters: space and tab
[:punct:] - punctuation characters
[:alnum:] - alphanumeric characters (alpha + digit)



```
❯ echo [[:upper:]]*
```

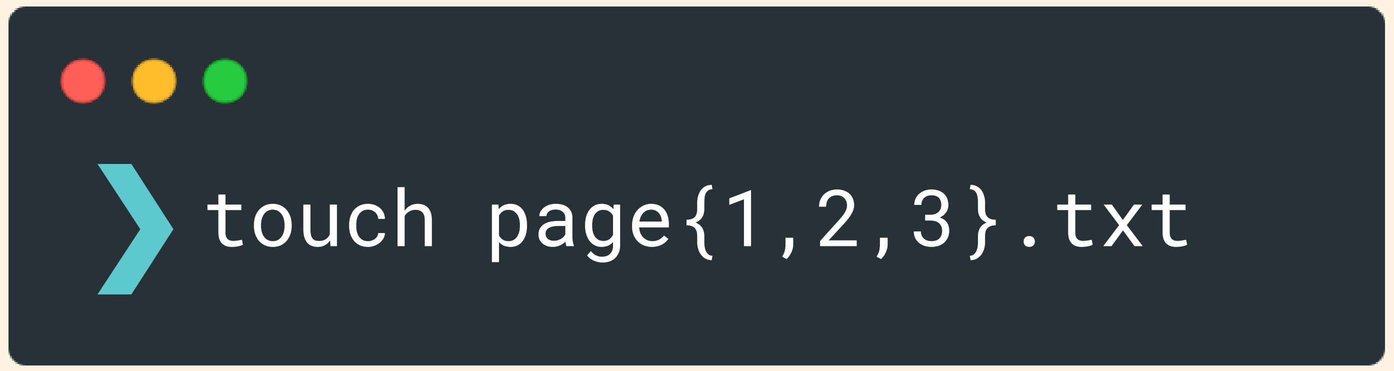
any file that starts with an uppercase letter

Brace Expansion

Brace expansion is used to generate arbitrary strings. Basically, it will generate multiple strings for us based on a pattern. We provide a set of strings inside of curly braces ({ }), as well as optional surrounding prefixes and suffixes.

The specified strings are used to generate all possible combinations with the optional prefixes and suffixes.

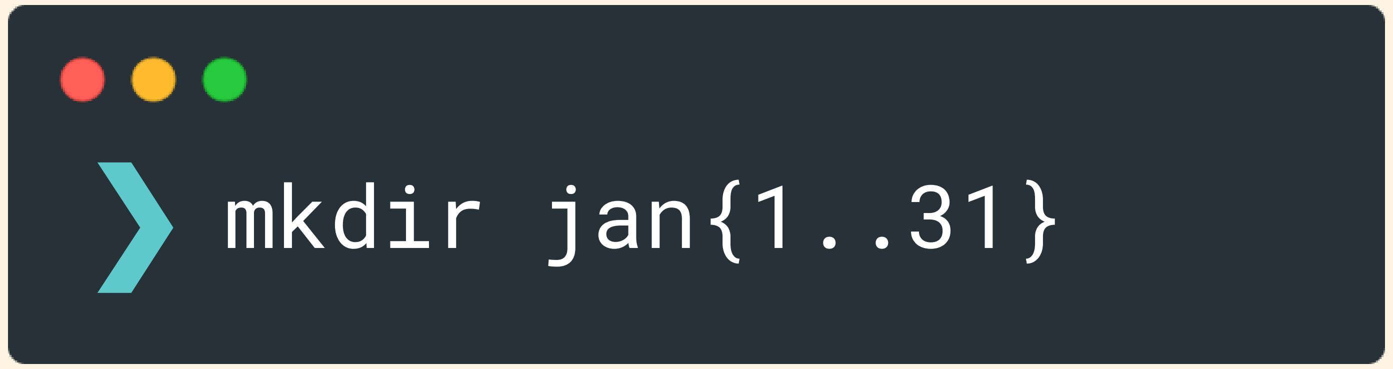
For example, touch page{1,2,3}.txt will generate three new files: page1.txt, page2.txt, and page3.txt



```
touch page{1,2,3}.txt
```

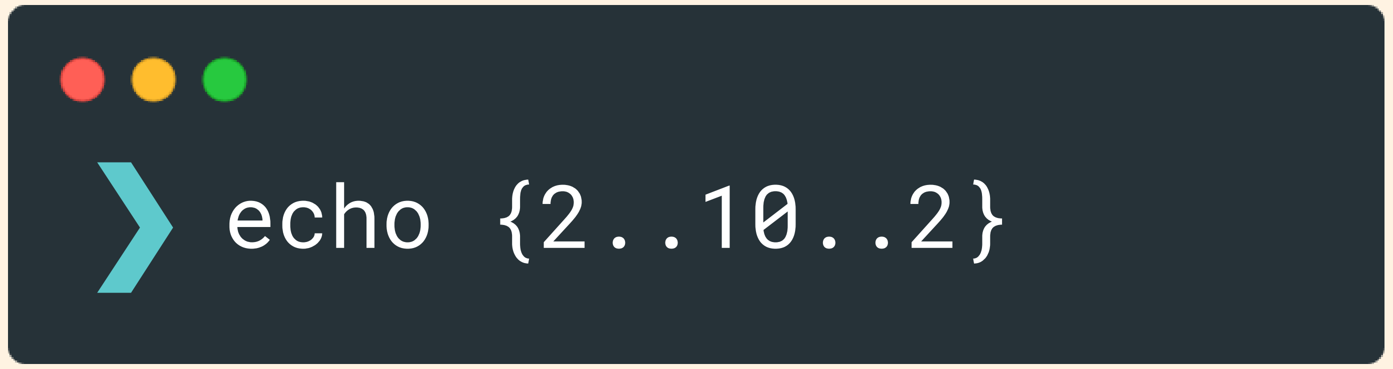
Ranges

We can provide a numeric range, which will be used to generate a sequence. In this example, jan{1..31} will be expanded to jan1, jan2, jan3, etc. until jan31.



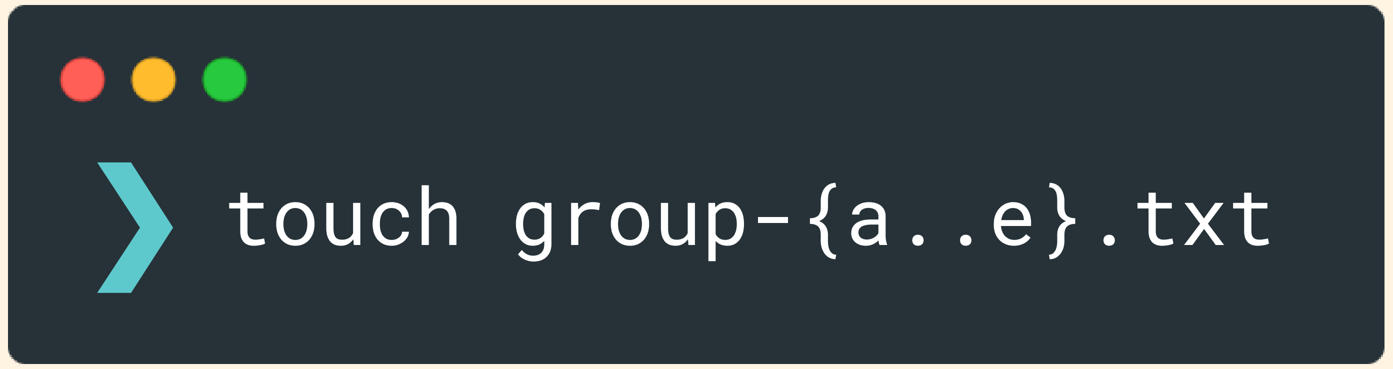
```
❯ mkdir jan{1..31}
```

We can provide a third value which defines the interval for the range. In this example, echo {2..10..2} will print out the numbers 2, 4, 6, 8, and 10



```
❯ echo {2..10..2}
```

We can even specify alphabetical ranges. This example generate the files group-a.txt, group-b.txt, group-c.txt,group-d.txt, and group-e.txt



```
❯ touch group-{a..e}.txt
```

Brace Expansion

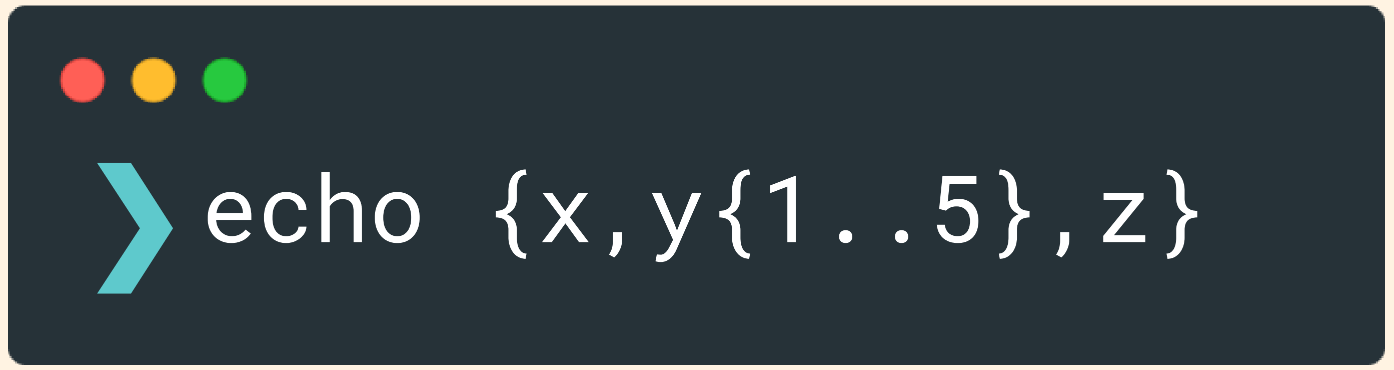
```
❯ echo {a,b,c}{1,2,3}
```

a1 a2 a3 b1 b2
b3 c1 c2 c3

```
❯ echo {b,r}{eef,at,ag}
```

beef bat bag
reef rat rag

Nested Brace Expansion



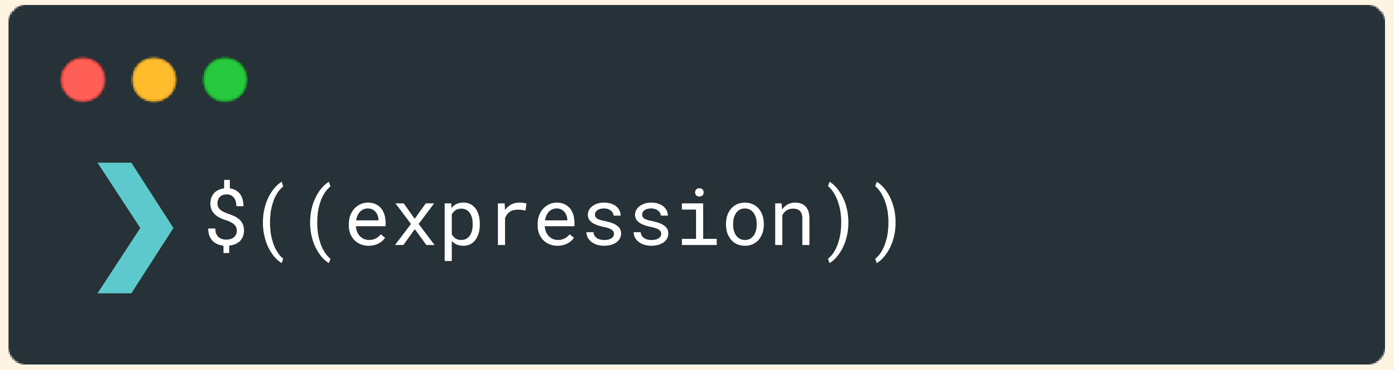
```
echo {x,y{1..5},z}
```

x y1 y2 y3 y4 y5 z

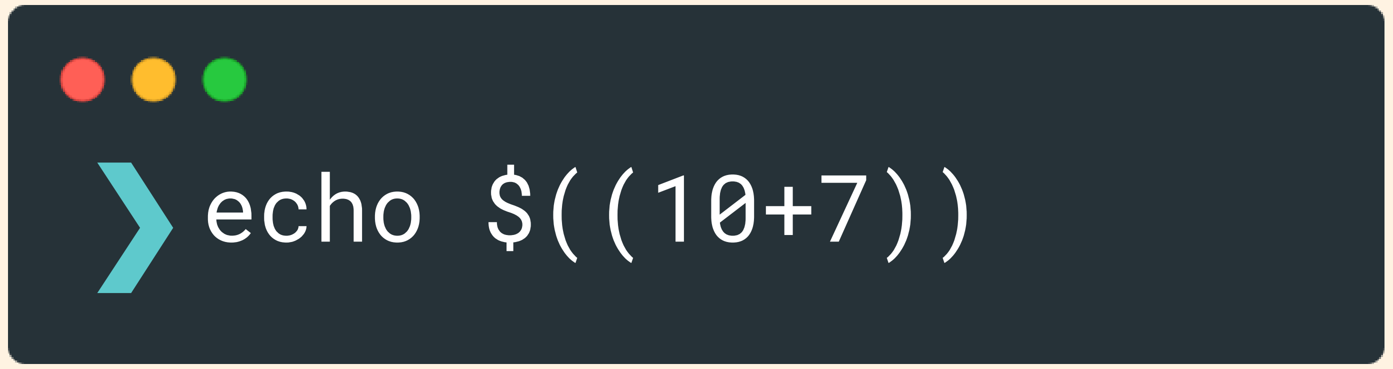
Arithmetic Expansion

The shell will perform arithmetic via expansion using the S((expression)) syntax. inside the parentheses, we can right artithmetic expressions using:

+ addition
- subtraction
* multiplication
/ division
** exponentiation
% modulo (remainder operator)

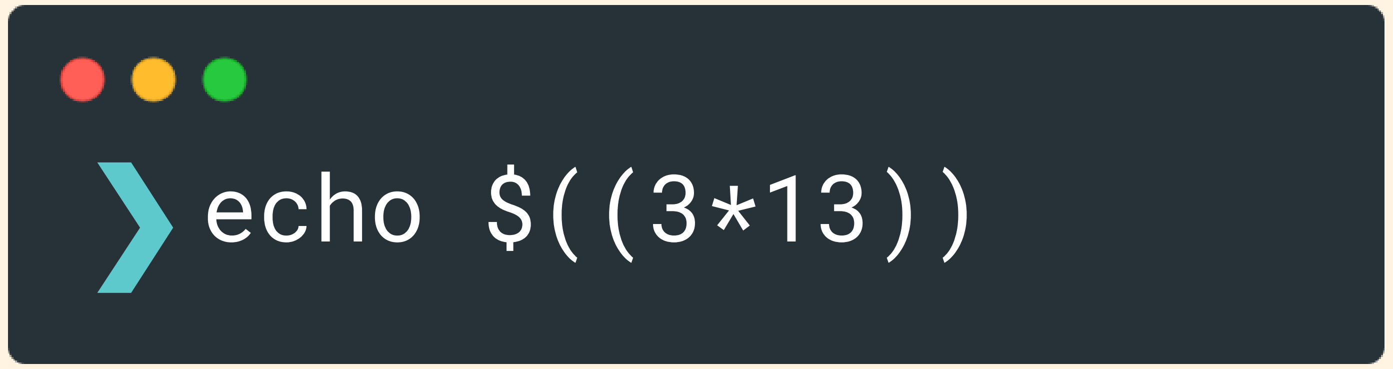


```
❯ $((expression))
```



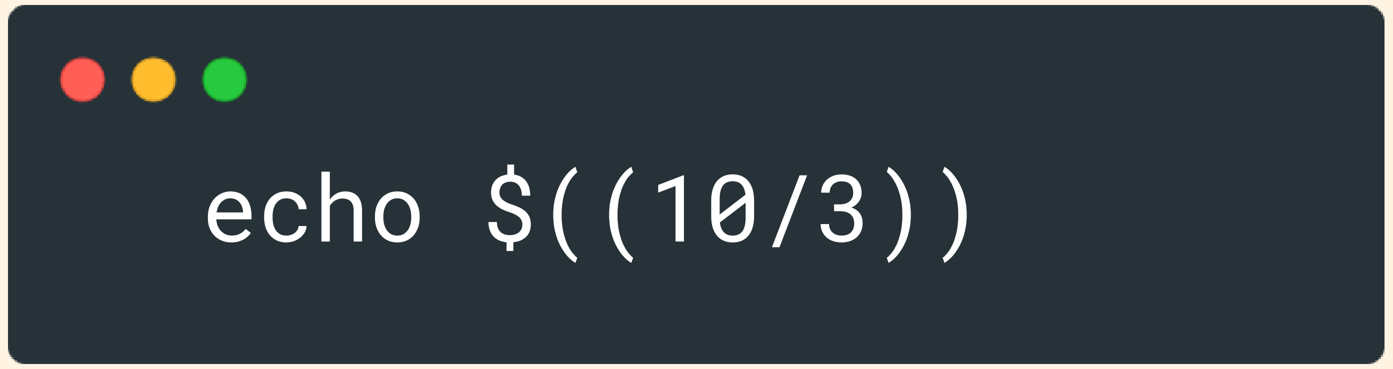
```
echo $((10+7))
```

17



```
echo $((3*13))
```

39



```
echo $((10/3))
```

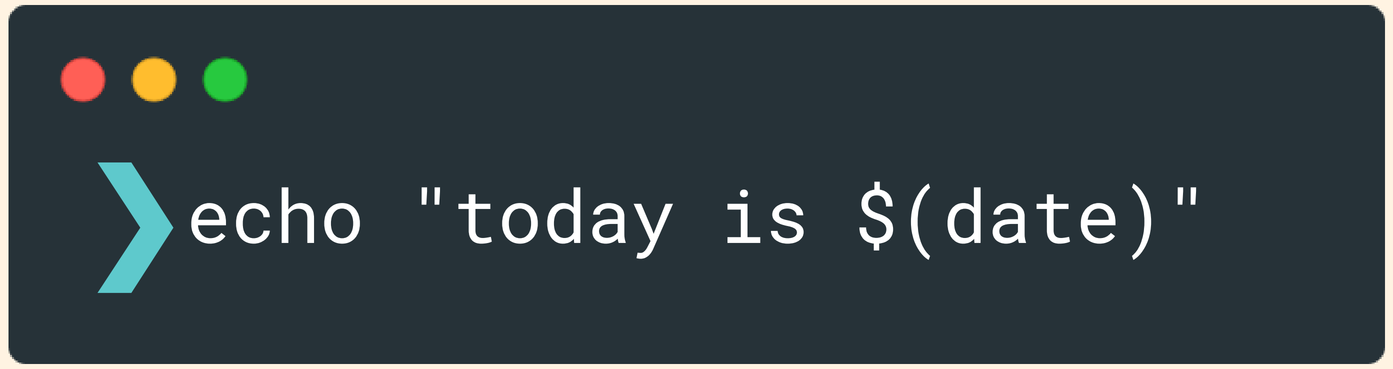
3

The shell only performs integer arithmetic, so the result is always a whole number

Command Substitution

We can use the $(command) syntax to display the output of another command.

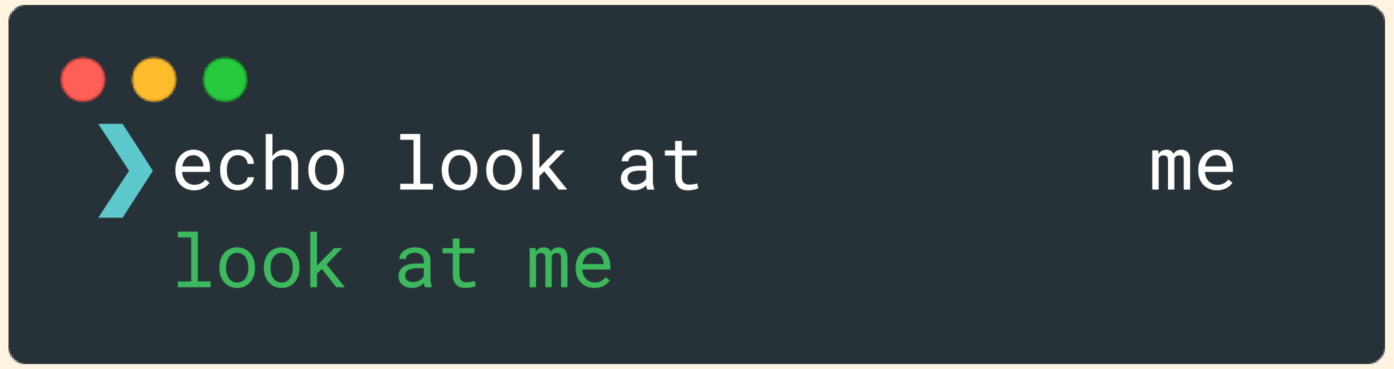
For example, echo "today is $(date)" will print "today is Thu 01 May 2021 03:10:31 PM PDT"



```
echo "today is $(date)"
```

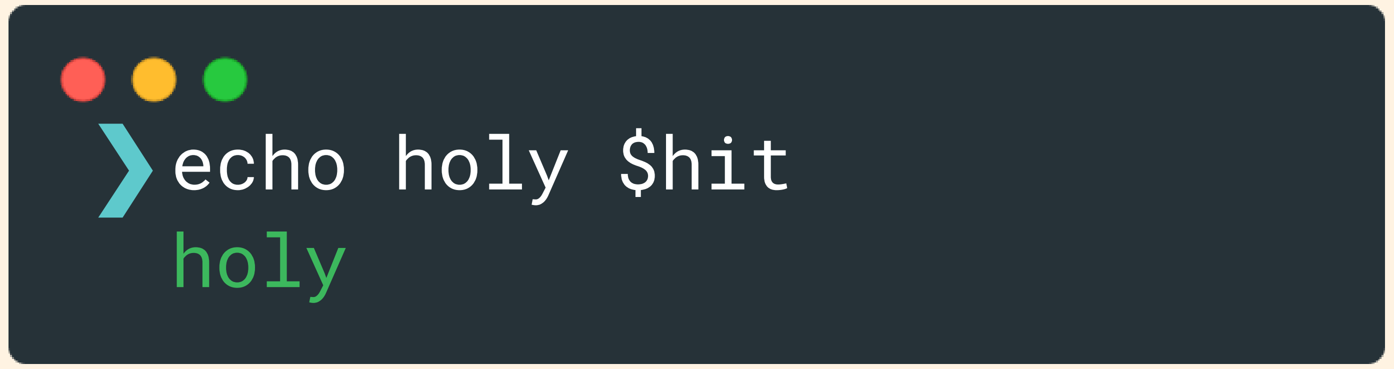
Quoting

In this example, our large whitespace is ignored because the shell performs something called word splitting.



```
❯echo look at          me
 look at me
```

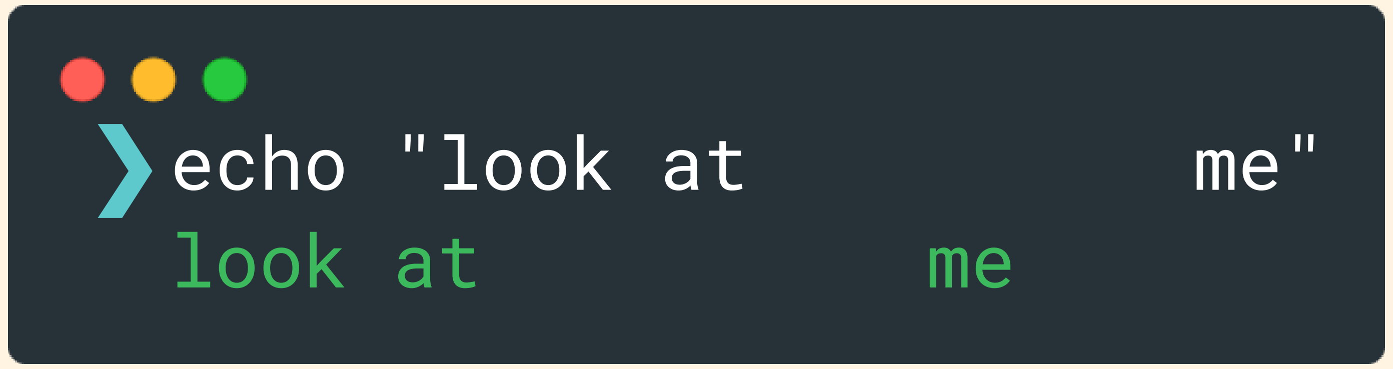
In this example, we only see "holy" printed out because the shell thinks we are referencing a variable called hit. It can't find a value for hit, so it substitutes an empty string instead.



```
❯echo holy $hit
 holy
```

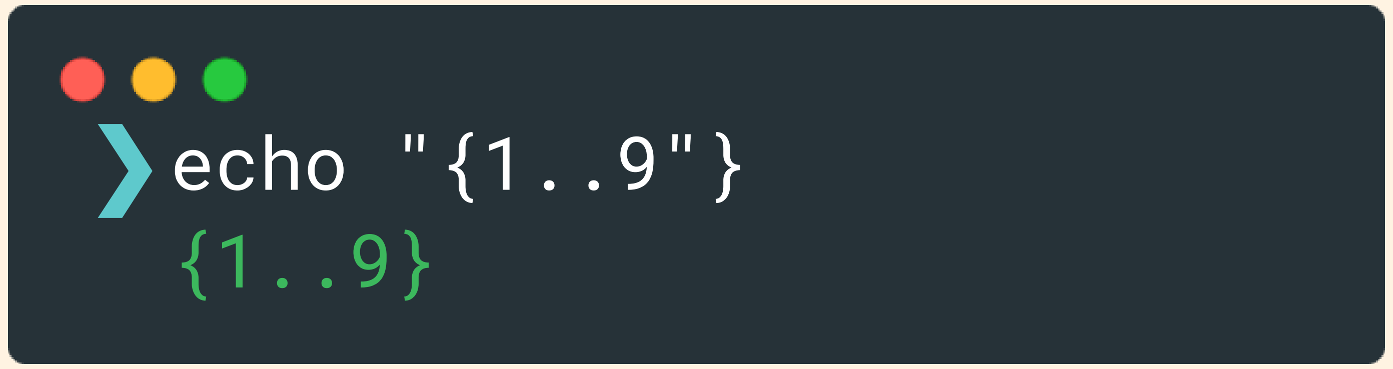
Double Quotes

If we wrap text in double quotes, the shell will respect our spacing and will ignore special characters except for dollar sign ($) , backslash (\ ), and backtick (`)



```
❯echo "look at           me"
 look at           me
```

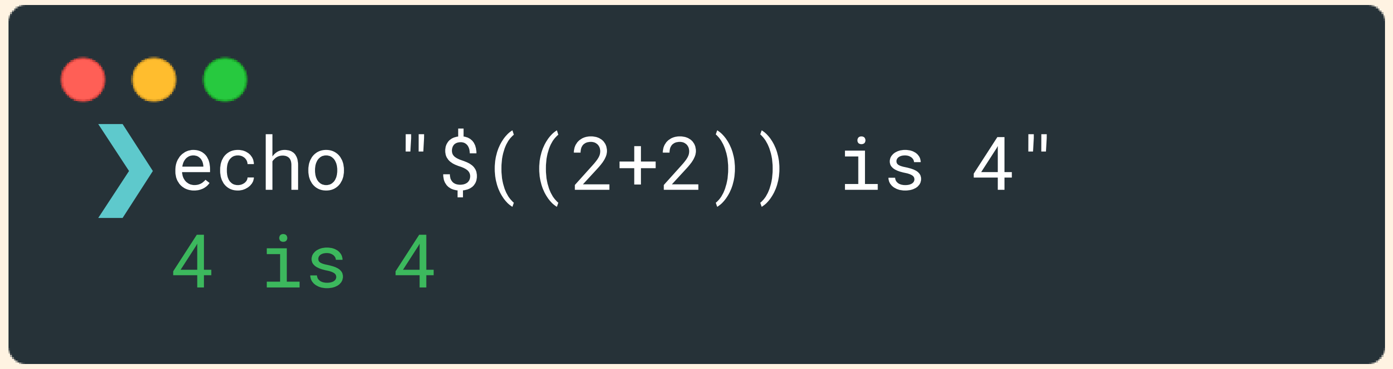
Pathname expansion, brace expansion, and word splitting will be ignored. However, command substitution and arithmetic expansion is still performed because dollar signs still have meaning inside double quotes.



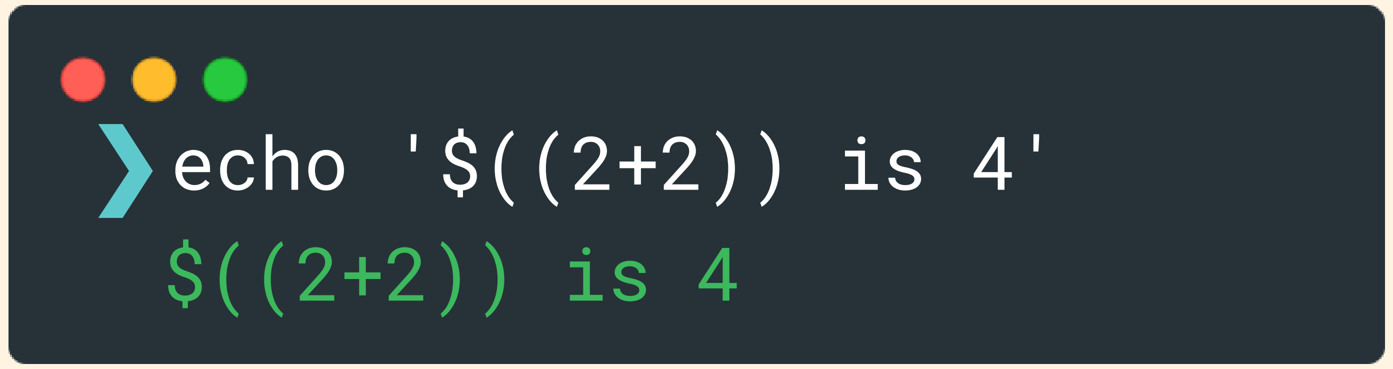
```
❯echo "{1..9"}
 {1..9}
```

Single Quotes

Use single quotes to suppress all forms of substitution.



```
❯echo "$((2+2)) is 4"
 4 is 4
```

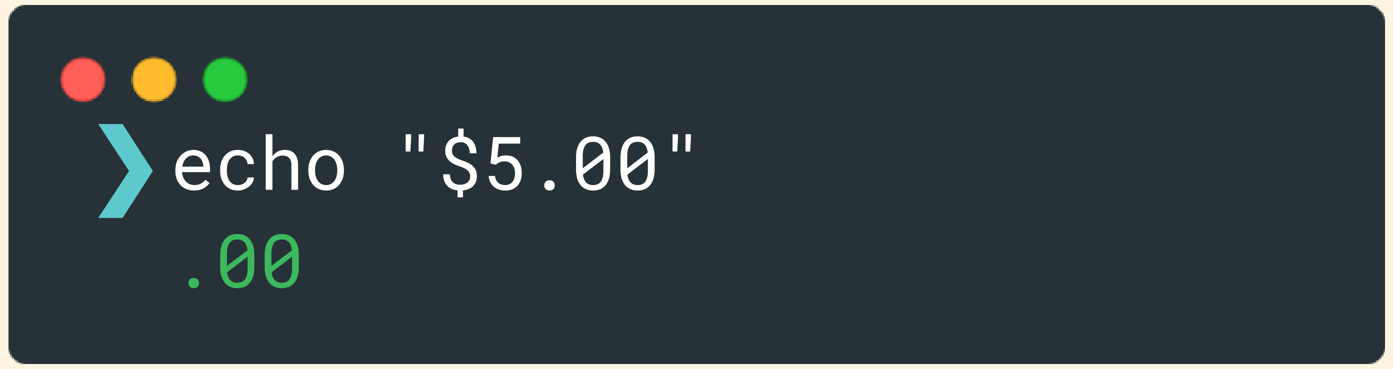


```
❯echo '$((2+2)) is 4'
 $((2+2)) is 4
```

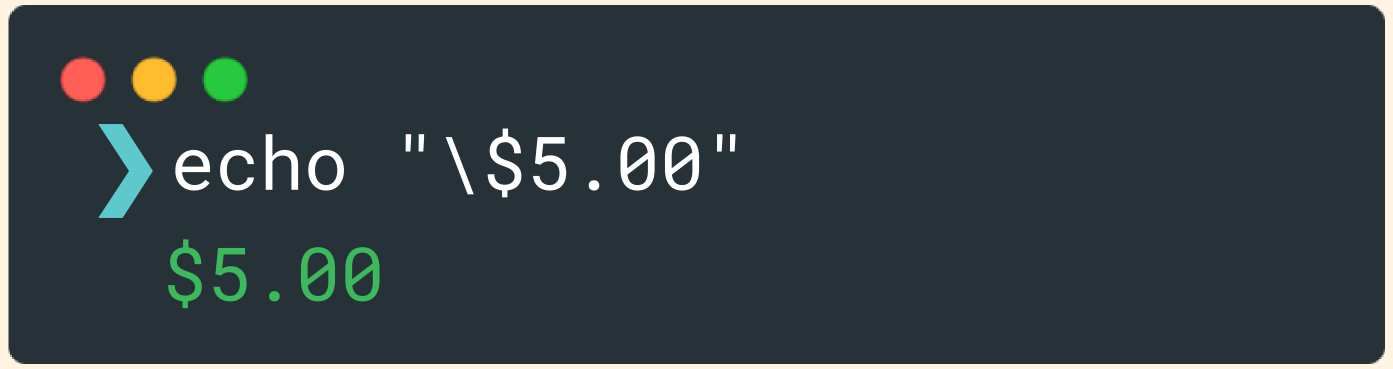
Escaping

To selectively prevent expansion or substitution for specific characters, we can prefix them with a single backslash.

We can use this to reference special characters that normally have meanings inside of filenames.



```
❯ echo "$5.00"
.00
```



```
❯ echo "\$5.00"
$5.00
```