A Bit Shifty

Bit hacks you should know

Primer on operators

```
&     bitwise AND            integers
|     bitwise OR             integers
^     bitwise XOR            integers
&^    bit clear (AND NOT)    integers

<<    left shift             integer << unsigned integer
>>    right shift            integer >> unsigned integer
```

Numbers in binary representation

```
fmt.Printf("%08b | %[1]d\n", i)
// 00001101 | 13
```

```
fmt.Printf("%08b | %[1]d\n", i)
// 00000000 | 0
```

Finding odd numbers

```
// the usual way
if 3 % 1 == 0 { /* odd */ }
```

```
// the bit hack way
if 3&1 == 1 { /* odd */ }
```

This works because...

```
    00101011 | 43
&   00000001 | 1
---------
00000001
```

Bit shifting

Observe which bit is set in this series

Operation	Binary	Integer
1 << 1	| 00000010 |	2
1 << 2	| 00000100 |	4
1 << 3	| 00001000 |	8
1 << 4	| 00010000 |	16
1 << 5	| 00100000 |	32
1 << 6	| 01000000 |	64
1 << 7	| 10000000 |	128

Turning on specific bits

```
// Given the number 120…
var n = 120
// → 01111000 | 120

// Now let’s turn on bit position 2
n = n | (1 << 2)
// → 01111100 | 124

// Can we turn it off now?
n = n &^ (1 << 2)
// → 01111000 | 120
```

Re-stating the obvious

- Think of a uint8 not as a number, but as 8 individual bits
- Each bit has 2 possible state, 1 or 0
- We now have 256 possible states

Bitwise Permissions

```
const (
        read    = 1 << iota // left shift by 0
        write               // left shift by 1
        update              // left shift by 2
        delete              // left shift by 3

        guest     = read
        user      = read | write | update
        moderator = read | delete
        admin     = read | write | update | delete
)
```

Note that read gives no permission.