hello world

main.go

```go
// TestDemo project main.go
package main

import (
    "fmt"
)

func main() {
    fmt.Println("Hello World!")
}
```

- import 导入需要的包
  import 声明必须跟在文件的 package 声明之后。随后，则是组成程序的函数、变量、常量、 类型的声明语句（分别由关键字 func , var , const , type 定义）。
- func 定义函数
  一个函数的声明由 func 关键字、函数名、参数列表、返回值列表（这个例子里的 main 函数 参数列表和返回值都是空的）以及包含在大括号里的函数体组成。

GO语言不需要在语句或者声明的末尾添加分号，除非一行上有多条语句。

命令行参数

- os包，提供一些与操作系统交互的函数和变量，os.Args程序的获取命令行参数
- os.Args的第一个元素，os.Args[0], 是命令本身的名字；其它的元素则是程序启动时传给它的 参数。s[m:n]形式的切片表达式，产生从第m个元素到第n-1个元素的**切片**
- echo命令实现

```go
// TestDemo project main.go
package main

import (
    "fmt"
    "os"
)

func main() {
    var s, sep string
    for i := 1; i < len(os.Args); i++ {
        S += SEP + os.Args[i]
        sep = " "
    }
    fmt.Println(s)
}
```

符号 := 是短变量声明（short variable declaration）的一部分, 这是定义一个或多个变量并根据它们的初始值为这些变量赋予适当类 型的语句。

- go语言的循环语句只有以下一种形式

```
for initialization; condition;  post    {              //  zero
or  more    statements }
```

//for循环三个部分不需括号包围。大括号强制要求, 左大括号必须和post语句在同一行。这三个部分都可以省略，就变成一个无限循环。

- 另一种循环方式

```
func main() {
    s, sep := "", ""
    for _, arg := range os.Args[1:] {
        s += sep + arg
        sep = " "
    }
    fmt.Println(s)
}
```

//_表示空标识符，用于任何语法需要变量名但是程序逻辑不需要的时候

- 变量的申明方式

```
s   :=  "" //只能用于函数内部，不能用于包变量
var s   string //默认初始化""
var s   =   "" //同时申明多个变量时用
var s   string  =   ""
//一般使用前两种方式
```

- 上述方式中的变量s将在适当时机对它gc，代价高昂，涉及的数据量很大时，推荐使用

```
(strings.Join(os.Args[1:],  "   ")
```

查找重复的行

main.go

```go
//寻找相邻的重复行
package main

import (
    "bufio"
    "fmt"
    "os"
)

func main() {
    counts := make(map[string]int)
    input := bufio.NewScanner(os.Stdin)
    for input.Scan() {
        counts[input.Text()]++
        for line, n := range counts {
            if n > 1 {
                fmt.Printf("%d\t%s\n", n, line)
            }
        }
    }
}
```

- map存储了键/值（key/value）的集合，例子中的键是字符串，值是整数，内置函数make 创建空 map

%d 十进制整数
%x, %o, %b 十六进制，八进制，二进制整数。
%f, %g, %e 浮点数： 3.141593 3.141592653589793 3.141593e+00
%t 布尔：true或false
%c 字符（rune） (Unicode码点)
%s 字符串
%q 带双引号的字符串”abc”或带单引号的字符’c’
%v 变量的自然形式（natural format）
%T 变量的类型
%% 字面上的百分号标志（无操作数）

起始页 | main.go | main.go | main.go

```go

import (
    "bufio"
    "fmt"
    "os"
)

func main() {
    counts := make(map[string]int)
    files := os.Args[1:]
    if len(files) == 0 {
        countLines(os.Stdin, counts)
    } else {
        for _, arg := range files {
            f, err := os.Open(arg)
            if err != nil { //nil相当于null,如果err不等于null，文件打开时出错
                fmt.Fprintf(os.Stderr, "dup2: %v\n", err) //出错时，用Fprintf与%v打印一条错误信息
                continue                                  //continue跳出
            }
            countLines(f, counts) //成功打开文件，读取文件，直到结束
            f.Close()             //关闭文件，释放资源
        }
    }
    for line, n := range counts {
        if n > 1 {
            fmt.Printf("%d\t%s\n", n, line)
        }
    }
}
func countLines(f *os.File, counts map[string]int) {
    input := bufio.NewScanner(f)
    for input.Scan() {
        counts[input.Text()]++
    }
}

```

起始页 | main.go | main.go | main.go

```go
// dup3 project main.go
package main

import (
    "fmt"
    "io/ioutil"
    "os"
    "strings"
)

func main() {
    counts := make(map[string]int)
    for _, filename := range os.Args[1:] {
        data, err := ioutil.ReadFile(filename) //返回的是一个字节切片，必须转化成string
        if err != nil {
            fmt.Fprintf(os.Stderr, "dup3:%v\n", err)
            continue
        }
        for _, line := range strings.Split(string(data), "\n") {
            counts[line]++
        }
    }
    for line, n := range counts {
        if n > 1 {
            fmt.Printf("%d\t%s\n", n, line)
        }
    }
}

```