

Copying Files

This lesson provides an implementation of copying files in Go.

How do you copy a file to another file? The simplest way is using `Copy` from the `io` package:





main.go

source.txt

```
package main
import (
    "fmt"
    "io"
    "os"
)

func main() {
    CopyFile("output/target.txt", "source.txt")
    fmt.Println("Copy done!")
}

func CopyFile(dstName, srcName string) (written int64, err error) {
    src, err := os.Open(srcName)
    if err != nil {
        return
    }
    defer src.Close()
    dst, err := os.OpenFile(dstName, os.O_WRONLY|os.O_CREATE, 0644)
    if err != nil {
        return
    }
    defer dst.Close()
    return io.Copy(dst, src)
}
```



Copy the Files

In the code above, we isolate the *copying* of the file in a function `CopyFile`, which takes as parameters first the *destination* file, and secondly the *source*

file. (see **line 9** where the function is called).

Look at the header of the `CopyFile` function at **line 13**. It simply opens the *source* file at **line 14**, handles an eventual error from **line 15** to **line 17** and assures the file is closed at the end of the function at **line 18**. In the same way, the *destination* file is opened at **line 19** for writing. An eventual error is handled from **line 20** to **line 22**, and it assures the file is closed at the end of the function at **line 18**. Then, at **line 24**, `io.Copy(dst, src)` is called and its values returned: the number of *bytes* written and a possible *error*. Note that, because of `defer`, the `src` and `dst` files are only closed after the *copy* is done and these values returned.

Note that by deferring `src.Close()` before the destination file is open ensures that, even if the opening of `dst` fails, `src.Close()` is still executed, closing the `src` file, which otherwise would remain open, using resources.

Now, you see how simple it is to copy content from one file to another in Go. In the next lesson, you'll learn how to read arguments from the command line.