Update from 4
Administrator создал 1 месяц назад

3d8eca41

README.md 7,4 КБ

Day 02 - Go Boot camp

Not Invented Here Syndrome

Contents

Chapter I

General rules

- Your programs should not quit unexpectedly (giving an error on a valid input). If this happens, your project will be considered non functional and will receive a 0 during the evaluation.
- We encourage you to create test programs for your project even though this work won't have to be submitted and won't be graded. It will give you a chance to easily test your work and your peers' work. You will find those tests especially useful during your defence. Indeed, during defence, you are free to use your tests and/or the tests of the peer you are evaluating.
- Submit your work to your assigned git repository. Only the work in the git repository will be graded.
- If your code is using external dependencies, it should use Go Modules for managing them

Chapter II

Rules of the day

- You should only turn in `*.go` files and (in case of external dependencies) `go.mod + go.sum`
- Your code for this task should be buildable with just `go build`

Chapter III

Intro

It's really amazing how much you can do just using command line utilities! Pretty much any OS, including embedded ones, has its own CLI and a set of small programs to do magical things. As an example, you can read about BusyBox, which is basically a swiss army knife for a variety of systems, starting with Linux-powered routers on OpenWRT and going to Android phones.

We're not trying to reinvent the wheel here, but knowing how to work with FS and perform basic CLI things in Golang can be really helpful, so let's spend some time on this.

Chapter IV

Exercise 00: Finding Things

As a first step, let's implement `find` -like utility using Go. It has to accept some path and a set of command-line options to be able to locate different types of entries. We are interested in three types of entries, which are directories, regular files and symbolic links. So, we should be able to run our program like this:

```
# Finding all files/directories/symlinks recursively in directory /foo
~$ ./myFind /foo
/foo/bar
```

```
/foo/bar/baz
/foo/bar/baz/deep/directory
/foo/bar/test.txt
/foo/bar/buzz -> /foo/bar/baz
/foo/bar/broken_sl -> [broken]
```

or specifying `-sl`, `-d` or `-f` to print only symlinks, only directories or only files. Keep in mind that user should be able to specify one, two or all three of them explicitly, like `./myFind -f -sl /path/to/dir` or `./myFind -d /path/to/other/dir`.

You should also implement one more option - `-ext` (works ONLY when -f is specified) for user to be able to print only files with a certain extension. An extension in this task can be thought of the last part of filename if we split it by a dot. So,

```
# Finding only *.go files ignoring all the rest.
~$ ./myFind -f -ext 'go' /go
/go/src/github.com/mycoolproject/main.go
/go/src/github.com/mycoolproject/magic.go
```

You'll also need to resolve symlinks. So, if `/foo/bar/buzz` is a symlink pointing to some other place in FS, like `/foo/bar/baz`, print both paths separated by `->`, like in example above.

Another thing about symlinks is that they may be broken (pointing to a non-existing file node). In this case your code should print `[broken]` instead of the path of a symlink destination.

Files and directories that current user doesn't have access to (permission errors) should be skipped in output and not lead to a runtime error.

Chapter V

Exercise 01: Counting Things

Now we are able to find our files, but we might need more meta information about what is in those files. Let's implement a `wc` -like utility to gather basic statistics about our files.

First things first, let's assume our files are utf-8 encoded text files, so your code should work with texts in Russian, too (forget about special cases like Arabic for now, only English and Russian are required). Also, you may ignore punctuation and just consider spaces as the only word delimiters.

You'll need to implement three mutually exclusive (only one can be specified at a time, otherwise an error message is printed) flags for your code: `-l` for counting lines, `-m` for counting characters and `-w` for counting words. Your program should be runnable like this:

```
# Counting words in file input.txt
~$ ./myWc -w input.txt
777 input.txt
# Counting lines in files input2.txt and input3.txt
~$ ./myWc -l input2.txt input3.txt
42 input2.txt
53 input3.txt
# Counting characters in files input4.txt, input5.txt and input6.txt
~$ ./myWc -m input4.txt input5.txt input6.txt
1337 input4.txt
2664 input5.txt
3991 input6.txt
```

As you may see, the answer is always a calculated number and a filename separated by tab (`\t`). If no flags are specified, `-w` behaviour should be used.

Important: as all files are independent, you should utilize goroutines to process them concurrently. You can start as many goroutines as there are input files specified for the program.

Chapter VI

Exercise 02: Running Things

Do you know what `xargs` is? You can read about it here, for example. Let's implement a similar tool - in this exercise you'll need to write a utility that will:

1. treat all parameters as a command, like 'wc -l' or 'ls -la'
2. build a command by appending all lines that are fed to program's stdin as this command's arguments, then execute it. So if we run

```
~$ echo -e "/a\n/b\n/c" | ./myXargs ls -la
```

it should be an equivalent to running

```
~$ ls -la /a /b /c
```

You can test this tool together with those from previous Exercises, so

```
~$ ./myFind -f -ext 'log' /path/to/some/logs | ./myXargs ./myWc -l
```

will calculate line counts for all ".log" files in `/path/to/some/logs` directory recursively.

Chapter VII

Exercise 03: Archiving Things

The last tool that we'll implement for this day is log rotation tool. "Log rotation" is a process when the old log file is archived and put away for storage so logs wouldn't pile up in a single file indefinitely. It should work like this:

```
# Will create file /path/to/logs/some_application_1600785299.tag.gz
# where 1600785299 is a UNIX timestamp made from `some_application.log`'s [MTIME](https://linuxize.com/post/linux-touch-command/)
~$ ./myRotate /path/to/logs/some_application.log
```

```
# Will create two tar.gz files with timestamps (one for every log)
# and put them into /data/archive directory
~$ ./myRotate -a /data/archive /path/to/logs/some_application.log /path/to/logs/other_application.log
```

As in Exercise 01, you should use goroutines to parallelize archiving of several files simultaneously.