

# What is Mining?

---

In this assignment you will implement proof-of-work mining. Before we get to the details on mining let's start at the beginning of our blockchain.

At the root of our blockchain is a special block called the "genesis" block. Basically the "genesis" block is the beginning block in our blockchain. It is special because it will not point back to any previous block.

The code that you are given can write out the genesis block, and an index to where to find it. In most blockchains like Bitcoin some sort of a database is used for storing the blocks.

We are not going to do that. We are going to store all of them in files in the file system. This is so you can see the blocks. Also we are going to store the blocks in JSON as text so you can read the blocks. Using a database is faster but has lots of overhead. It can also be very frustrating when you are attempting to determine if the block is correct and you can't easily see what is in the block.

Our blocks will be written (by default) in the `./data` directory. The format for the blocks is `hash.json`, where hash is the block hash.

To get started, first checkout the code for Assignment 2 - this can be done by: (You can go online to it and cut-paste the link - that is what I usually do. In the browser go to <https://github.com/Univ-Wyo-Education>. Then click on the `S19-4010` repository. When that comes up there is a green button on the left that says `Clone or download`. Click on that. Cut and paste the URL.)

(You have probably already done this)

```
cd ~/go/src/github.com/  
mkdir Univ-Wyo-Education  
cd Univ-Wyo-Education  
git clone https://github.com/Univ-Wyo-Education/S19-4010.git  
cd S19-4010.git
```

If you have already cloned the S19-4010 set of information you will need to update it to current. You may have made changes already.

```
cd ~/go/src/github.com/Univ-Wyo-Education/S19-4010  
git stash  
git pull  
git checkout -b hw02
```

The `checkout -b` will create a new branch for you to work on.

Having your own branch (you can name it other than hw2) will allow you to switch back and forth between the original code and your modified code.

Then from the `~/go/src/github.com/Univ-Wyo-Education/S19-4010` directory change directory into assignment 2.

```
cd a/02
```

Our starting code is in this directory. Specifically we will want to compile the main program. It is in `./main`. Cd to that directory. You should end up in: `~/go/src/github.com/Univ-Wyo-Education/S19-4010/a/02`

```
cd ./main
go get
go build
```

`go get` will pull in any dependent packages that are needed to build this.

Run main to create the genesis block and the initial index.

```
./main --create-genesis
```

This should create a directory with 2 files in it. The default is in the `./data` directory (`.../a/02/main/data`). The files are:

```
136c53391115ab7ff717bd24e62dd0df2c270500d7194290169a83488022548e.json
index.json
```

You should look at the contents of the 2 files. Edit the files with a text editor like `vi`, `vim` or whatever editor you use for editing code (don't modify the files). The one with the long name is our genesis block. The `index.json` is an index that will allow us to find data blocks as we are building the this blockchain.

The code is missing the chunk that will do the block mining. The stubbed out function is in `.../a/02/mine/mine.go`. Your assignment is to implement the body of the function. You will want to verify that it works by running

```
go test
```

in the `.../a/02/mine` directory. If you run that now you should get `FAIL` because you have not implemented it yet.

Take the time to go and poke through the code. This code is the basis for your mid-term project. You are going to need to be familiar with all of it. Run all the tests. If you have questions about it now is the time to be asking them.

## Mining

Mining is the process of doing some hard work that anybody can easily check to verify that a digital seal has been put on a block. The seals that we will use are hashes with special properties. In our case the property will be that the first 4 characters of the hex string representation of the hash will need to be zeros. "0000" at the beginning of the hash. To generate a hash with this pattern we will include a 64 bit integer in the data. Each time we hash the data if we do not get a hash with our special property we will increment the integer and try again. After enough increments we will stumble upon a hash with the properly.

The difficulty is controlled by the number of 0's at the beginning of our hash. If we want to increase the difficulty we can go to 00000 or 000000 zeros. In Bitcoin this difficulty automatically increases. In Ethereum this difficulty is set by group consensus.

### Pseudo Code for the mining.

Code that I have supplied you with:

go Package	Description
block	Operations on blocks like initialization and serialization.
	Look in the <code>.../a/02/block/block.go</code> file.
hash	Convenience functions to work with keccak256 hash.
	Look in the <code>.../a/02/hash/hash.go</code> file.

go Library functions you will need to use:

go Package	Description
hex	Convert from/to base 16 strings.
	<a href="https://golang.org/pkg/encoding/hex/">https://golang.org/pkg/encoding/hex/</a>
fmt	Generate formatted output.
	<a href="https://golang.org/pkg/fmt">https://golang.org/pkg/fmt</a>

In the file `./mine/mine.go`, implement the function `MineBlock`.

1. Use an infinite loop to:
  1. Serialize the data from the block for hashing, Call `block.SerializeForSeal` to do this.
  2. Calculate the hash of the data, Call `hash.Hash0f` to do this. This is the slow part. What would happen if we replaced the software with a hash calculator on a graphics card where you could run 4096 hashes at once? What would happen if we replaced the graphics card with an ASIC - so you had dedicated hardware to do the hash and you could run 4 billion hashes a second?
  3. Convert the hash (it is `[]byte`) to a hex string. Use the `hex.EncodeToString` standard go library function.
  4. `fmt.Printf("((Mining)) Hash for Block [%s] nonce [%8d]\n", theHashAsString, bk.Nonce)`
  5. See if the first 4 characters of the hash are 0's. - if so we have met the work criteria. In go this is `if theHashAsString[0:4] == "0000" {`. This is create a slice, 4 long from character 0 with length of 4, then compare that to the string `"0000"`.
  6. Set the block's "Seal" to the hash
  7. `fmt.Printf("((Mining)) Hash for Block [%s] nonce [%8d]\n", theHashAsString, bk.Nonce)`
  8. `return`
  9. Increment the Nonce in the block, and...
  10. Back to the top of the loop for another try at finding a seal for this block.

In the `./mine` directory there is a test. Implement the `MineBlock` function. Run the test. Remove the `InstructorImplementationMineBlock(bk) // TODO: Replace this line with your code.` Put your code in place of it.

Submit your completed `./mine/mine.go` file.

My output when running the test.

```
+> go test
((Mining)) Hash for Block [0000ae2cab130b4836988969f731c4f884ac4675790e5575a516
((Mining)) Hash for Block [0000adc29a80f1f0df08c8687c013d179050f5d1b449599e4d14
((Mining)) Hash for Block [000013ce557332aaa68abe3b7bf1be856743a03689a802606a73
PASS
ok      github.com/Univ-Wyo-Education/S19-4010/a/02/mine      0.914s
```

The grader has a somewhat more comprehensive automated test to run with this code (There is one more block to mine). You get all the points for the assignment when it passes the test.

## Before you submit your code!

Use the go formatter on your code. Either `goimports -w *.go` or setup your editor to run `goimports` every time you save a go file. I have this setup in `vim`. Other editors can do this also.

Run `go vet` and `golint *.go` on it. Fix any errors.

## Submit

1. mine.go
2. a copy of your output from running `go test` in the `./mine` directory.