**Automate your smart contracts actions - Makefile**

**The magic of Makefile**

You are a hero for getting this far! If you think a bit about your experience with the whole FundMe project by now, how many times have you written a forge script NameOfScript --rpc-url xyz --private-key 0xPrivateKey .... There's got to be an easier way to run scripts and other commands.

The answer for all your troubles is a Makefile!

A Makefile is a special file used in conjunction with the make command in Unix-based systems and some other environments. It provides instructions for automating the process of building software projects.

The main advantages of using a Makefile are:

* Automates tasks related to building and deploying your smart contracts.
* Integrates with Foundry commands like forge build, forge test and forge script.
* Can manage dependencies between different smart contract files.
* Streamlines the development workflow by reducing repetitive manual commands.
* Allows you to automatically grab the .env contents.

In the root folder of your project create a new file called Makefile.

After creating the file run make in your terminal.

If you have make installed then you should receive the following message:

make: \*\*\* No targets. Stop

If you don't get this message you need to install make. This is a perfect time to ask your favorite AI to help, but if you still don't manage it please come on the Updraft section of Cyfrin discord and ask the lovely people there.

Let's start our Makefile with -include .env on the first line. This way we don't have to call source .env every time we want to access something from it.

Soo... how do we actually write a shortcut?

Let's write one for forge build.

In your Makefile write the following line:

build:; forge build

Run make build in your terminal.

make build

forge build

[⠔] Compiling...

No files changed, compilation skipped

And it works! We've written our first shortcut. Arguably not the best of shortcuts, we've saved 1 letter, but still, it's a start.

**Small note**: The :; between build and forge build is used to indicate that the command will be given on the same line. Change the build shortcut as follows:

build:

forge build

Run it again.

Let's write a more complex shortcut. Add the following shortcut to your Makefile:

deploy-sepolia:

forge script script/DeployFundMe.s.sol:DeployFundMe --rpc-url $(SEPOLIA\_RPC\_URL) --private-key $(SEPOLIA\_PRIVATE\_KEY) --broadcast --verify --etherscan-api-key $(ETHERSCAN\_API\_KEY) -vvvv

Now this is a mouthful. But you already know what we did above, we used forge script to deploy fundMe on Sepolia, using the private key and rpc-url we provided in the .env, we used --broadcast for the transaction to be broadcasted, and then we verify the contract on etherscan. The only difference compared to what we did before is encapsulating .env variables between round brackets. This is how Makefile knows these come from the .env.

The --verify option verifies all the contracts found in the receipts of a script, if any. We could use the --verifier option to select another verifier, but we don't need that because the default option is etherscan. So the only thing we need is an etherscan API key. To get one go to [Etherscan.io](https://etherscan.io/register) and make an account. After that, log in, go to OTHERS > API Keys add a new project and copy the API Key Token.

Open your .env file and add the following line:

ETHERSCAN\_API\_KEY=THEAPIKEYYOUCOPIEDFROMETHERSCANGOESHERE

Make sure your .env holds all the things it needs to run the shortcut above. Again, we do not use private keys associated with accounts that hold real money. Stay safe!

The moment of truth:

make deploy-sepolia

ONCHAIN EXECUTION COMPLETE & SUCCESSFUL.

Total Paid: 0.004665908735630779 ETH (577031 gas \* avg 8.086062509 gwei)

##

Start verification for (1) contracts

Start verifying contract `0x2BC3f6eB5C38532F70DD59AC6A0610453bc16e9f` deployed on sepolia

Submitting verification for [src/FundMe.sol:FundMe] 0x2BC3f6eB5C38532F70DD59AC6A0610453bc16e9f.

Submitting verification for [src/FundMe.sol:FundMe] 0x2BC3f6eB5C38532F70DD59AC6A0610453bc16e9f.

Submitting verification for [src/FundMe.sol:FundMe] 0x2BC3f6eB5C38532F70DD59AC6A0610453bc16e9f.

Submitting verification for [src/FundMe.sol:FundMe] 0x2BC3f6eB5C38532F70DD59AC6A0610453bc16e9f.

Submitted contract for verification:

Response: `OK`

GUID: `cjgaycqnrssgths7jakwgbexwjzpa5tirhymzvhkrxitznnvzx`

URL: https://sepolia.etherscan.io/address/0x2bc3f6eb5c38532f70dd59ac6a0610453bc16e9f

Contract verification status:

Response: `NOTOK`

Details: `Pending in queue`

Contract verification status:

Response: `OK`

Details: `Pass - Verified`

Contract successfully verified

All (1) contracts were verified!

The contract is deployed on Sepolia and we verified it on [Etherscan](https://sepolia.etherscan.io/address/0x2bc3f6eb5c38532f70dd59ac6a0610453bc16e9f).

Amazing work!

This is just an introductory lesson on how to write Makefiles. Properly organizing your scripts and then transforming them into shortcuts that save you from typing 3 lines of code in the terminal is an ART!

Let's pass through some examples. Go copy the [Makefile available in the Fund Me repo](https://github.com/Cyfrin/foundry-fund-me-f23/blob/main/Makefile).

Treat this Makefile as a framework for your projects.

Open the file and go through it.

The .PHONY: tells make that all the all test clean deploy fund help install snapshot format anvil are not folders. Following that we declare the DEFAULT\_ANVIL\_KEY and a custom help message.

Run make help to print it in your terminal.

There are a lot of useful shortcuts related to dependencies, formatting, deployment etc.

For example, run the following commands:

make anvil

Open a new terminal.

make deploy

And you just deployed a fresh FundMe contract on a fresh anvil blockchain. Super-fast and super cool!

We could do the same for Sepolia by running make deploy ARGS="--network sepolia".

Makefile is amazing!