Foundry Notes

Deploying with Anvil

Simulation ->

```
forge script script/DeployContract.sol --rpc-url <ANVIL CHAIN URL>
```

Actual deployment on local chain ->

```
forge script script/DeployContract.sol --rpc-url <LOCAL CHAIN URL> --
broadcast <PRIVATE KEY from ANVIL CHAIN>
```

Importing from GitHub

To import a library from GitHub, we need to run the following command:

```
forge install <OWNER-NAME>/<REPO-NAME>@<VERSION> --no-commit
# For example
forge install smartcontractkit/chainlink-brownie-contracts@1.1.1 --no-
commit
```

Encrypting Private Key

Run `cast wallet import -i <name>` or `cast wallet import <name> --interactive`.
Paste the private key into the prompt.
Set a strong password at least 20 chars long so that it can't be brute-forced.
Use the account and sender flag when running Foundry commands like this: `forge script <script_location>:<contract_name> --rpc-url <rpc_url> --account <name> --sender <public_key> --broadcast`.
If actually deploying to, for example, Sepolia, we would also add `--verify --etherscan-api-key`.
Type `cast wallet list` to check the list of wallets.
(Can also go to home directory, then `cd ./foundry/keystores/`, then `ls` to see list of keystores.)
Then type `cast <wallet_name>` to see the encrypted version of the private key which follows ERC-2335.

Call and send with Command-line

Run `cast send <Contract_Address> "<fn. sign>" args --rpc-url <RPC_URL> --private-key <PRIVATE_KEY>` to send values to a function as parameters. Sending transaction.
`cast call` has the same arguments as above but it is used for calling a function where a value is returned. Calling transaction.
`cast --to-base <HEX_VALUE> dec` to convert hex values to decimal values.

zkSync

Setup

foundry-zksync installed.
To switch back to vanilla Foundry, simply run `foundryup`, and to switch back to foundry-zksync, run `foundryup-zksync`
Run `forge build --zksync` to compile with zkSync

Foundry Commands

- `forge test`
  - `--match-test` for specifying a test function. `-m` is deprecated.
  - `--match-path` for specifying the path of the test contract.
  - `--fork-url` for forking any network.
  - `--gas-report` for gas report.
- `forge snapshot`
  - Creates a file (`.gas-snapshot`) with the gas costs of the test
  - Use with same commands as for `forge test`
- `forge fmt`
  - To format your code
- `forge inspect`
  - `<CONTRACT NAME> storageLayout` shows the storage of the contract
  - `constant` and `immutable` global variables do not show up in storage
- `forge coverage`
  - `--report debug` gives the details of all tests and the lines that aren't covered by the tests. Adding `> <FILE_NAME>.txt` to it would store the output in a txt file in the directory.

Foundry Cheatcodes

- `vm.expectRevert(...)` is used when the next LoC is supposed to revert. If not, the test fails.
- `vm.prank(address)` sets the provided address as the msg.sender for the next call.
- `makeAddr(string)` takes a name as a string and generates an address for the same name.
- `vm.deal(address, uint256)` takes an address and gives it an amount of tokens
- `hoax(address, uint256)` combination of `vm.prank()` and `vm.deal()`
- `vm.txGasPrice(uint256)` sets the tx.gasprice() for the rest of the transaction
- `vm.warp(uint256)` alters block.timestamp
- `vm.expectEmit(bool,bool,bool,bool, address)` checks if an event was emitted. The first three bool values are for indexed values emitted with events, and the fourth one is for any non-indexed values present in the events. The address is of the contract.
- `vm.roll(uint256)` alters block.number

- `vm.recordLogs()` records all the emitted events
- `vm.getRecordedLogs()` gets all the recorded logs in an array of Log objects, defined in `forge-std/Vm.sol`

Foundry Tools

- `Cyfrin/foundry-devops`
    - `./src/DevOpsTools.sol` - Can be used to get the most recent deployment of your contract. Need to set `ffi = true` in `foundry.toml` so that foundry can run bash scripts on your device, preferred to keep it as `false` in general.

Random Notes

> - Gas costs can be calculated by taking gas used in testnet, multiply by latest gas price on mainnet and convert to USD. Visible that Eth mainnet is very expensive so prefer to deploy on an L2 chain like zkSync.
> - forge -> Compiling and interacting with contracts
> - cast -> Interacting with contracts that have already been deployed
> - anvil -> To deploy a local blockchain
> - chisel -> To type and run small snippets of solidity in terminal, maybe for checking something or testing
> - `address` cannot be explicitly cast as `uint256`. It needs to be cast as `uint160` and then as `uint256`.
>
> ```
> address a = msg.sender;
> return uint256(uint160(a));
> ```
>
> - When deploying with anvil and using it, the gas price defaults to 0.