**Calculate Withdraw gas costs**

**About that Gas everyone is so passionate about**

Gas refers to a unit that measures the computational effort required to execute a specific operation on the network. You can think of it as a fee you pay to miners or validators for processing your transaction and storing the data on the blockchain.

An important aspect of smart contract development is making your code efficient to minimize the gas you/other users spend when calling functions. This can have a serious impact on user retention for your protocol. Imagine you have to exchange 0.1 ETH for 300 USDC, but you have to pay 30 USDC in gas fees. No one wants to pay that. It's your duty as a developer to minimize gas consumption.

Now that you understand the importance of minimizing gas consumption, how do we find out how much gas things cost?

Let's take a closer look at testWithdrawFromASingleFunder test.

Run the following command in your terminal:

forge snapshot --mt testWithdrawFromASingleFunder

You'll see that a new file appeared in your project root folder: .gas-snapshot. When you open it you'll find the following:

FundMeTest:testWithdrawFromASingleFunder() (gas: 84824)

This means that calling that test function consumes 84824 gas. How do we find out what this means in $?

Etherscan provides a super nice tool that we can use: <https://etherscan.io/gastracker>. Here, at the moment I'm writing this lesson, it says that the gas price is around 7 gwei. If we multiply the two it gives us a total price of 593,768 gwei. Ok, at least that's an amount we can work with. Now we will use the handy [Alchemy converter](https://www.alchemy.com/gwei-calculator) to find out that 593,768 gwei = 0.000593768 ETH and 1 ETH = 2.975,59 USD according to [Coinmarketcap](https://coinmarketcap.com/) meaning that our transaction would cost 1.77 USD on Ethereum mainnet. Let's see if we can lower this.

**Note: The gas price and ETH price illustrated above correspond to the date and time this lesson was written, these vary, please use the links presented above to find out the current gas and ETH price**

Looking closer at the testWithdrawFromASingleFunder one can observe that we found out the initial balances, then we called a transaction and then we asserted that startingFundMeBalance + startingOwnerBalance matches the expected balance, but inside that test we called withdraw which should have cost gas. Why didn't the gas we paid affect our balances? Simple, for testing purposes the Anvil gas price is defaulted to 0 (different from what we talked about above in the case of Ethereum mainnet where the gas price was around 7 gwei), so it wouldn't interfere with our testing.

Let's change that and force the withdraw transaction to have a gas price.

At the top of your FundMeTest contract define the following variable:

uint256 constant GAS\_PRICE = 1;

and refactor the testWithdrawFromASingleFunder function as follows:

function testWithdrawFromASingleFunder() public funded {

// Arrange

uint256 startingFundMeBalance = address(fundMe).balance;

uint256 startingOwnerBalance = fundMe.getOwner().balance;

vm.txGasPrice(GAS\_PRICE);

uint256 gasStart = gasleft();

// Act

vm.startPrank(fundMe.getOwner());

fundMe.withdraw();

vm.stopPrank();

uint256 gasEnd = gasleft();

uint256 gasUsed = (gasStart - gasEnd) \* tx.gasprice;

console.log("Withdraw consumed: %d gas", gasUsed);

// Assert

uint256 endingFundMeBalance = address(fundMe).balance;

uint256 endingOwnerBalance = fundMe.getOwner().balance;

assertEq(endingFundMeBalance, 0);

assertEq(

startingFundMeBalance + startingOwnerBalance,

endingOwnerBalance

);

}

We changed the following:

1. We used a new cheatcode called vm.txGasPrice, this sets up the transaction gas price for the next transaction. Read more about it [here](https://book.getfoundry.sh/cheatcodes/tx-gas-price).
2. We used gasleft() to find out how much gas we had before and after we called the transaction. Then we subtracted them to find out how much the withdraw transaction consumed. gasleft() is a built-in Solidity function that returns the amount of gas remaining in the current Ethereum transaction.
3. Then we logged the consumed gas. Read more about [Console Logging here](https://book.getfoundry.sh/reference/forge-std/console-log)

Let's run the test:

forge test --mt testWithdrawFromASingleFunder -vv

[⠰] Compiling...

[⠔] Compiling 26 files with Solc 0.8.19

[⠘] Solc 0.8.19 finished in 1.06s

Compiler run successful!

Ran 1 test for test/FundMe.t.sol:FundMeTest

[PASS] testWithdrawFromASingleFunder() (gas: 87869)

Logs:

Withdraw consummed: 10628 gas

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.67ms (2.06ms CPU time)

Cool!

Now that we learned how to calculate the amount of gas used, let's learn how to make it better!