BLOCK NONCE #7: Snapshot Testing - Without Making Everyone Angry

What?

Snapshot tests are tests that contain hard-coded values that rarely change. For example, the gas cost of a fixed transaction is expected to change only if the compiler version is updated.

Why?

A dev that breaks a snapshot test is usually not inclined to read the broken test just to see how to fix it.

How?

Use the `expect-test` crate, to hard-code values that are auto-fixable! On failure, the command to fix the snapshot values is printed to console.
Instead of `assert_eq!(computed_value, 7)`, do:

```
expect!["7"].assert_debug_eq(computed_value);
```

Or, if the expected output is in a file (and not inlined in code), do:

```
expect_file!["path_to_file.txt"].assert_debug_eq(computed_value);
```

For comparing large objects, consider the specific snapshotted field separately. For example, instead of:

```
let expected_tx_result = <HARD CODED TX RESULT>;
let tx_result = run_tx();
assert_eq!(tx_result, expected_tx_result);
```

Do:

```
let mut expected_tx_result = <HARD CODED TX RESULT>;
let mut tx_result = run_tx();
expect!["7"].assert_debug_eq(tx_result.gas_cost);
// Overwrite the single snapshot field.
expected_tx_result.gas_cost = 0;
tx_result.gas_cost = 0;
assert_eq!(tx_result, expected_tx_result);
```

TIP💡 Remember to take different test cases into account! Maybe one `expect!` is not enough...

```
fn my_gas_test(#[values(2, 3)] gas_price: u8) {
    let result = execute_tx(gas_price).gas_price;
    if gas_price == 2 {
        expect!["18"].assert_debug_eq(result);
    } else {
        expect!["27"].assert_debug_eq(result);
    }
}
```