THE DEFINITIVE GUIDE TO TESTING SMART CONTRACTS ON STELLAR

LEIGH MCCULLOCH

Principal Software Engineer,
Stellar Development Foundation


____leigh___
____leigh___
leighmcculloch

DOES IT WORK?

“

A METHOD FOR ACQUIRING KNOWLEDGE THAT...

—Wikipedia “scientific method” (CC BY-SA 4.0)

“

...INVOLVES CAREFUL OBSERVATION COUPLED WITH RIGOROUS SCEPTICISM

—Wikipedia “scientific method” (CC BY-SA 4.0)

“

...COGNITIVE ASSUMPTIONS CAN DISTORT THE INTERPRETATION OF OBSERVATION

—Wikipedia “scientific method” (CC BY-SA 4.0)

ASSUMPTIONS

“

IF DEBUGGING IS THE PROCESS OF REMOVING SOFTWARE BUGS...

—Edsger W. Dijkstra

“

THEN PROGRAMMING MUST BE THE PROCESS OF PUTTING THEM IN.

—Edsger W. Dijkstra

“

THEN PROGRAMMING MUST BE THE PROCESS OF PUTTING THEM IN.

—Edsger W. Dijkstra

“

THEN PROGRAMMING MUST BE THE PROCESS OF PUTTING THEM IN.

—Edsger W. Dijkstra

TEST

TESTING STELLAR CONTRACTS

ONE LANGAGE

TESTING STELLAR CONTRACTS

ONE LANGAGE

REDUCE
CONTEXT
SWITCHING

TESTING STELLAR CONTRACTS

ONE LANGAGE

TESTING STELLAR CONTRACTS

REDUCE CONTEXT SWITCHING

CONSISTENT WITH MAINNET

ONE LANGAGE

TESTING STELLAR CONTRACTS

REDUCE CONTEXT SWITCHING

CONSISTENT WITH MAINNET

ADVANCED TESTING

ONE LANGAGE

TESTING STELLAR CONTRACTS

REDUCE CONTEXT SWITCHING

PROPERTY TESTING

FUZZING

CONSISTENT WITH MAINNET

ADVANCED TESTING

ONE LANGAGE

TESTING STELLAR CONTRACTS

BATTERIES INCLUDED

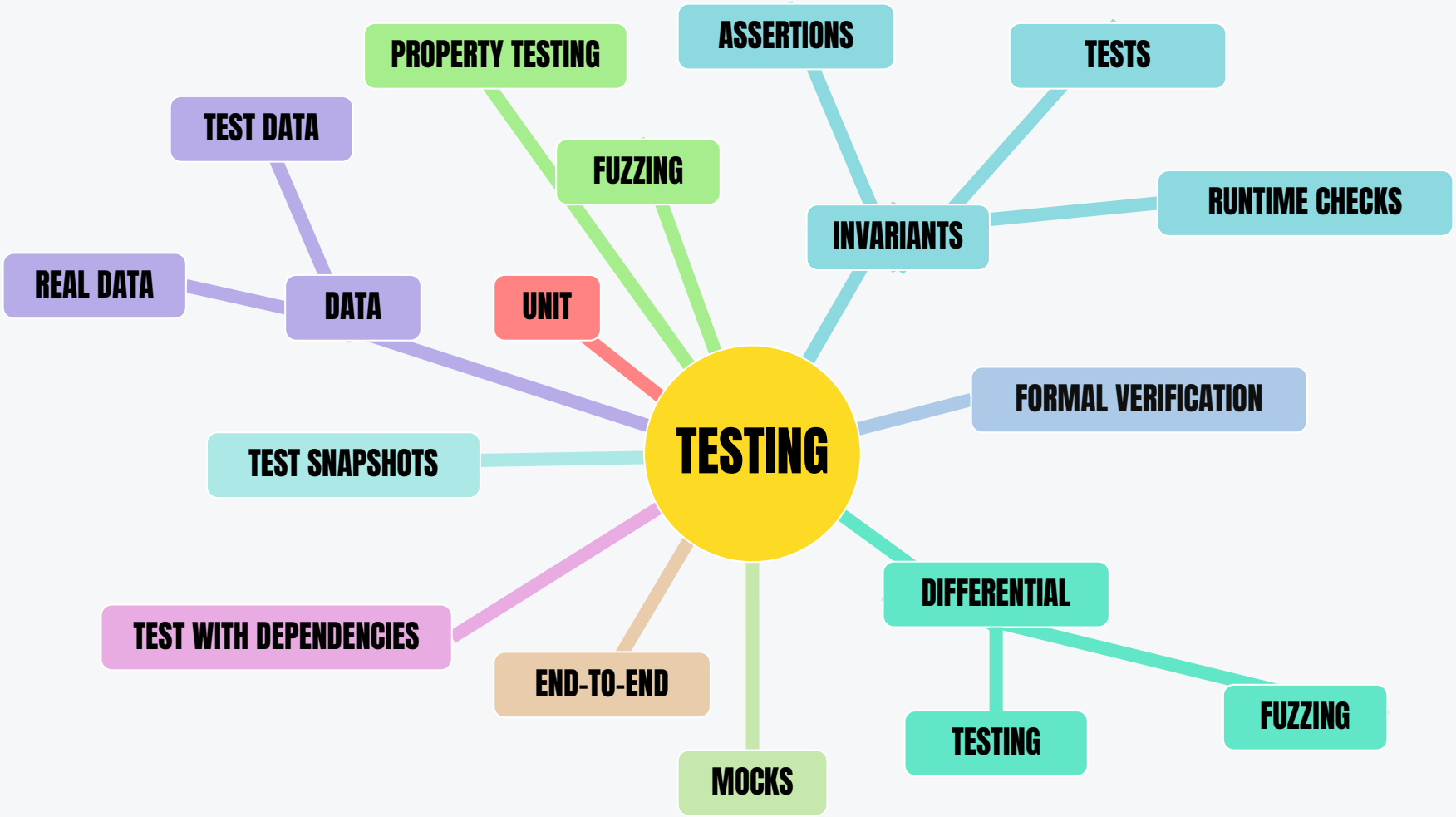
PROPERTY TESTING

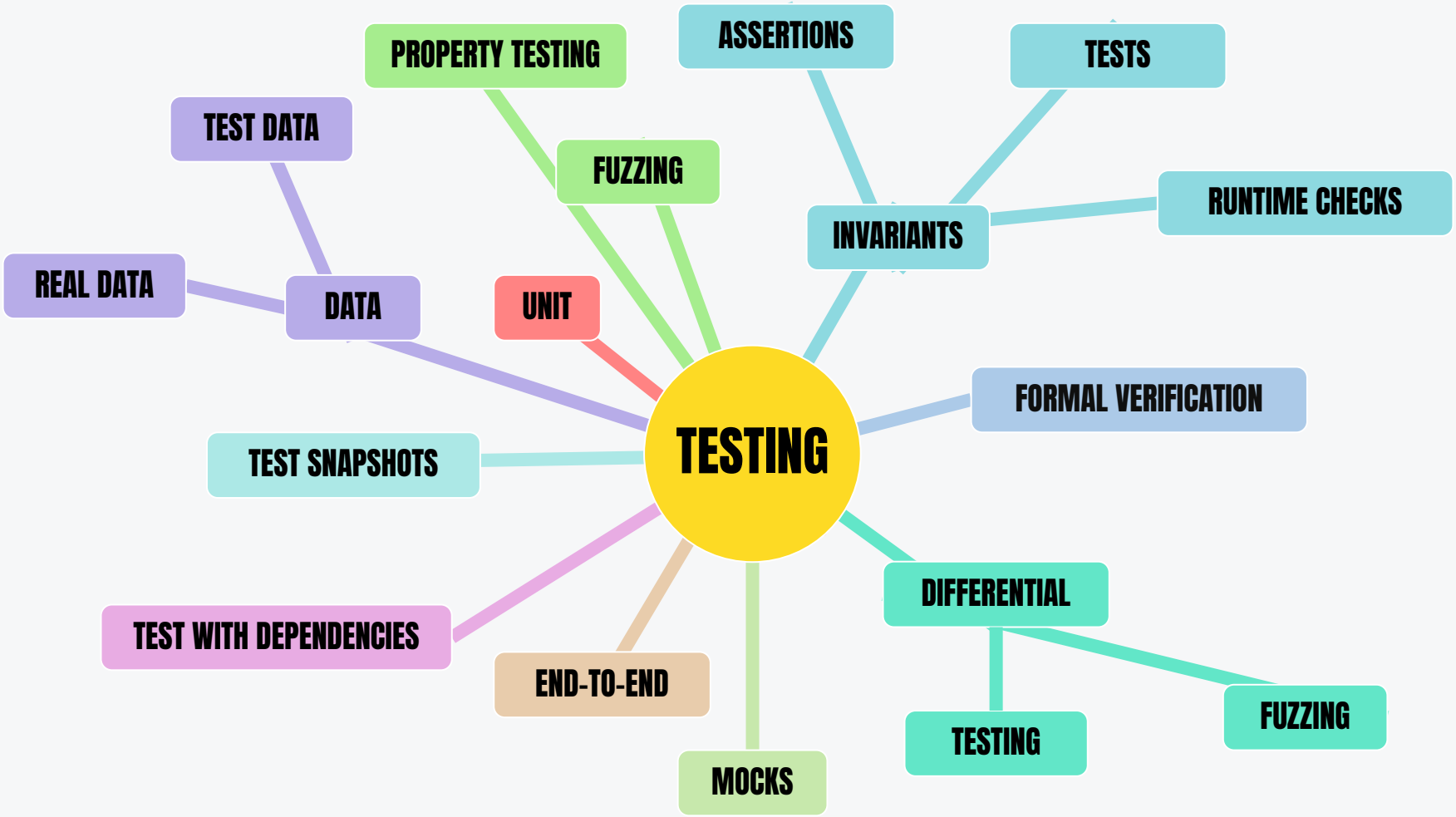
FUZZING

REDUCE CONTEXT SWITCHING

CONSISTENT WITH MAINNET

ADVANCED TESTING


TESTING
PROPERTY TESTING
ASSERTIONS
TESTS
TEST DATA
FUZZING
INVARIANTS
RUNTIME CHECKS
REAL DATA
DATA
UNIT
FORMAL VERIFICATION
TEST SNAPSHOTS
DIFFERENTIAL
TEST WITH DEPENDENCIES
END-TO-END
FUZZING
TESTING
MOCKS


TESTING
PROPERTY TESTING
ASSERTIONS
TESTS
TEST DATA
FUZZING
INVARIANTS
RUNTIME CHECKS
REAL DATA
DATA
UNIT
FORMAL VERIFICATION
TEST SNAPSHOTS
DIFFERENTIAL
TEST WITH DEPENDENCIES
END-TO-END
FUZZING
TESTING
MOCKS

developers.stellar.org

```rust
#[contractimpl]
impl Token {
    pub fn __constructor(admin: Address)

    pub fn mint(to: Address, amount: i128) → Result<(), Error>

    pub fn balance(addr: Address) → i128

    pub fn transfer(from: Address, to: Address, amount: i128)
        → Result<(), Error>
}

#[contracterror]
pub enum Error {
    Overflow = 1,
    InsufficientBalance = 2,
    NegativeAmount = 3,
}
```

CONTRACT

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

YOU + YOUR IDE = GOOD TIMES

```rust
#[test]
fn test() {
    let env = Env::default();

    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    assert_eq!(token.balance(&a), 0);
}
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    let b = Address::generate(&env);
    token.mock_all_auths().mint(&a, &10);
    assert_eq!(token.balance(&a), 10);
    assert_eq!(token.balance(&b), 0);

    token.mock_all_auths().transfer(&a, &b, &2);
    // ... assert on auths
    assert_eq!(token.balance(&a), 8);
    assert_eq!(token.balance(&b), 2);
    // ... assert on events
```

TEST

```rust
#[test]
fn test() {
    let env = Env::default();
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    let b = Address::generate(&env);
    token.mock_all_auths().mint(&a, &10);
    assert_eq!(token.balance(&a), 10);
    assert_eq!(token.balance(&b), 0);

    token.mock_all_auths().transfer(&a, &b, &2);
    // ... assert on auths
    assert_eq!(token.balance(&a), 8);
    assert_eq!(token.balance(&b), 2);
    // ... assert on events
```

TEST

```rust
token.mock_all_auths().transfer(&a, &b, &2);

assert_eq!(env.auths(), [
    (
        a.clone(),
        AuthorizedInvocation {
            function: AuthorizedFunction::Contract((
                token.address.clone(),
                symbol_short!("transfer"),
                (&a, &b, 2i128).into_val(&env),
            )),
            sub_invocations: [].into(),
        }
    ),
]);

assert_eq!(token.balance(&a), 8);
assert_eq!(token.balance(&b), 2);
```

TEST

```rust
assert_eq!(token.balance(&a), 8);
assert_eq!(token.balance(&b), 2);

assert_eq!(
    env.events().all(),
    vec![
        &env,
        (
            token.address.clone(),
            (symbol_short!("transfer"), &a, &b).into_val(&env),
            2i128.into_val(&env),
        )
    ],
);
```

TEST

```rust
#[contractimpl]
impl Token {
    pub fn __constructor(admin: Address, pause: Address)

    pub fn mint(to: Address, amount: i128) → Result<(), Error>

    pub fn balance(addr: Address) → i128

    pub fn transfer(from: Address, to: Address, amount: i128)
        → Result<(), Error>
}

#[contracterror]
pub enum Error {
    Overflow = 1,
    InsufficientBalance = 2,
    NegativeAmount = 3,
    Paused = 4,
}
```

CONTRACT

TEST WITH MOCK

```rust
#[contract]
struct Pause;
#[contractimpl]
impl Pause {
    pub fn paused() -> bool { false }
}

#[test]
fn test_not_paused() {
    let env = Env::default();
    let admin = Address::generate(&env);
    let pause = env.register(Pause, ());
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

TEST WITH MOCK

```rust
#[contract]
struct Pause;
#[contractimpl]
impl Pause {
    pub fn paused() → bool { false }
}

#[test]
fn test_not_paused() {
    // test setup
    token.mock_all_auths().mint(&a, &10);
    token.mock_all_auths().transfer(&a, &b, &2);
    // assert auths
    assert_eq!(token.balance(&a), 8);
    assert_eq!(token.balance(&b), 2);
    // assert events
}
```

TEST WITH MOCK

```rust
#[contract]
struct Pause;
#[contractimpl]
impl Pause {
    pub fn paused() → bool { true }
}

#[test]
fn test_paused() {
    // test setup
    token.mock_all_auths().mint(&a, &10);
    assert_eq!(
        token.mock_all_auths().try_transfer(&a, &b, &2),
        Err(Ok(Error::Paused))
    );
}
```

TEST WITH MOCK

```rust
#[contract]
struct Pause;
#[contractimpl]
impl Pause {
    pub fn paused() → bool { true }
}

#[test]
fn test_paused() {
    // test setup
    token.mock_all_auths().mint(&a, &10);
    assert_eq!(
        token.mock_all_auths().try_transfer(&a, &b, &2),
        Err(Ok(Error::Paused))
    );
}
```

TEST WITH MOCK

```rust
#[contract]
struct Pause;
#[contractimpl]
impl Pause {
    pub fn paused() -> ? { ? }
}

#[test]
fn test_?() {
    // test setup
    token.mock_all_auths().mint(&a, &10);
    assert_eq!(
        token.mock_all_auths().try_transfer(&a, &b, &2),
        ?
    );
}
```

TEST WITH MOCK

```rust
#[contract]
struct Pause;
#[contractimpl]
impl Pause {
    pub fn paused() -> ? { ? }
}

#[test]
fn test_?() {
    // test setup
    token.mock_all_auths().mint(&a, &10);
    assert_eq!(
        token.mock_all_auths().try_transfer(&a, &b, &2),
        ?
    );
}
```

A SOFTWARE TESTER WALKS INTO A BAR

A SOFTWARE TESTER WALKS INTO A BAR

AND ORDERS

A SOFTWARE TESTER WALKS INTO A BAR
AND ORDERS

1 DRINK

A SOFTWARE TESTER WALKS INTO A BAR
AND ORDERS
1 DRINK
2 DRINKS

A SOFTWARE TESTER WALKS INTO A BAR
AND ORDERS
1 DRINK
2 DRINKS
0 DRINKS

A SOFTWARE TESTER WALKS INTO A BAR
AND ORDERS
1 BEER
2 BEERS
0 BEERS
99999999 DRINKS

AND ORDERS

1 DRINK

2 DRINKS

0 DRINKS

99999999 DRINKS

A LIZARD IN A GLASS

1 DRINK

2 DRINKS

0 DRINKS

99999999 DRINKS

A LIZARD IN A GLASS

-1 DRINK

2 DRINKS

0 DRINKS

99999999 DRINKS

A LIZARD IN A GLASS

-1 DRINK

“QWERTYUIOP” DRINKS

0 DRINKS
99999999 DRINKS
A LIZARD IN A GLASS
-1 DRINK
“QWERTYUIOP” DRINKS
TESTING COMPLETE

99999999 DRINKS
A LIZARD IN A GLASS
-1 DRINK
“QWERTYUIOP” DRINKS
TESTING COMPLETE
A REAL CUSTOMER WALKS INTO THE BAR AND ASKS

A LIZARD IN A GLASS
-1 DRINK
“QWERTYUIOP” DRINKS
TESTING COMPLETE
A REAL CUSTOMER WALKS INTO THE BAR AND ASKS
WHERE IS THE BATHROOM?

-1 DRINK

“QWERTYUIOP” DRINKS

TESTING COMPLETE

A REAL CUSTOMER WALKS INTO THE BAR AND ASKS WHERE IS THE BATHROOM?

THE BAR GOES UP IN FLAMES

UNIT TESTING IS NOT ENOUGH

1. TESTING WITH REAL DEPENDENCIES
2. TESTING WITH REAL DATA
3. FUZZING AND PROPERTY TESTING
4. INVARIANTS
5. DIFFERENTIAL TESTING

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

TESTING WITH REAL DEPENDENCIES

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

```rust
mod pause {
    soroban_sdk::contractimport!(file = "pause.wasm");
}

#[test]
fn test() {
    let env = Env::default();
    let pause = env.register(pause::WASM, ());

    // ...
}
```

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

```rust
mod pause {
    soroban_sdk::contractimport!(file = "pause.wasm");
}

#[test]
fn test() {
    let env = Env::default();
    let pause = env.register(pause::WASM, ());
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

1

```rust
mod pause {
    soroban_sdk::contractimport!(file = "pause.wasm");
}

#[test]
fn test() {
    let env = Env::default();
    let pause = env.register(pause::WASM, ());
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

1

```rust
mod pause {
    soroban_sdk::contractimport!(file = "pause.wasm");
}

#[test]
fn test() {
    let env = Env::default();
    let pause = env.register(pause::WASM, ());
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

2

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

1

```rust
mod pause {
    soroban_sdk::contractimport!(file = "pause.wasm");
}

#[test]
fn test() {
    let env = Env::default();
    let pause = env.register(pause::WASM, ());
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

2

3

TESTING WITH REAL DEPENDENCIES

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

```rust
mod pause {
    soroban_sdk::contractimport!(file = "pause.wasm");
}

#[test]
fn test() {
    let env = Env::default();
    let pause = env.register(pause::WASM, ());
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

TESTING WITH REAL DATA

TESTING WITH REAL DATA

```
$ stellar snapshot create --address C...  --out snapshot.json
```

TESTING WITH REAL DATA

TESTING WITH REAL DATA

```
$ stellar snapshot create --address C... --out snapshot.json
```

```rust
#[test]
fn test() {
    let env = Env::from_ledger_snapshot("snapshot.json");

    // ...
}
```

```
$ stellar snapshot create --address C... --out snapshot.json
```

TESTING WITH REAL DATA

```rust
#[test]
fn test() {
    let env = Env::from_ledger_snapshot("snapshot.json");
    let admin = Address::generate(&env);
    let pause = Address::from_str(&env, "C...");
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

```
$ stellar snapshot create --address C... --out snapshot.json
```

TESTING WITH REAL DATA

```rust
#[test]
fn test() {
    let env = Env::from_ledger_snapshot("snapshot.json");
    let admin = Address::generate(&env);
    let pause = Address::from_str(&env, "C...");
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

```
$ stellar snapshot create --address C... --out snapshot.json
```

TESTING WITH REAL DATA

```rust
#[test]
fn test() {
    let env = Env::from_ledger_snapshot("snapshot.json");
    let admin = Address::generate(&env);
    let pause = Address::from_str(&env, "C...");
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

```
$ stellar snapshot create --address C... --out snapshot.json
```

TESTING WITH REAL DATA

```rust
#[test]
fn test() {
    let env = Env::from_ledger_snapshot("snapshot.json");
    let admin = Address::generate(&env);
    let pause = Address::from_str(&env, "C...");
    let id = env.register(Token, (&admin, &pause));
    let token = TokenClient::new(&env, &id);

    // ...
}
```

FUZZING

FUZZING AND PROPERTY TESTING

FUZZING

FUZZING AND PROPERTY TESTING

FUZZING

FUZZING AND PROPERTY TESTING

FUZZING

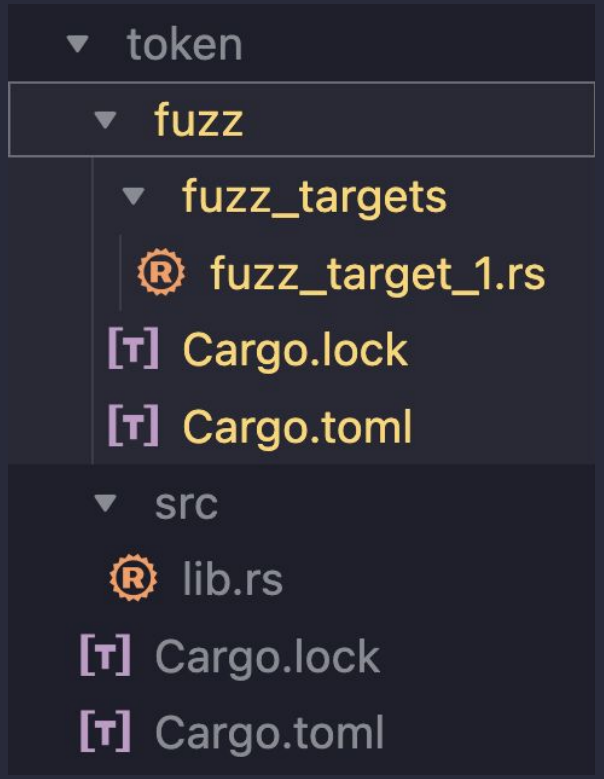
FUZZING AND PROPERTY TESTING

FUZZING

FUZZING AND PROPERTY TESTING

```
$ cargo install --locked cargo-fuzz
```

```
$ cargo fuzz init
```



```
$ cargo fuzz init
```

```
fuzz_target!(|input: Input| {
    // ...
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```
fuzz_target!(|input: Input| {
    // ...
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```
fuzz_target!(|input: Input| {
    // ...
});
```

```
#[derive(Arbitrary)]
pub struct Input {
    pub a: i128,
    pub b: i128,
    pub amount: i128,
}
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {
    let env = Env::default();
    let admin = Address::generate(&env);
    let id = env.register(Token, (&admin,));
    let token = TokenClient::new(&env, &id);

    let a = Address::generate(&env);
    let b = Address::generate(&env);
    _ = token.mock_all_auths().try_mint(&a, &input.a);
    _ = token.mock_all_auths().try_mint(&b, &input.b);

    // ...
```

```rust
#[derive(Arbitrary)]
pub struct Input {
    pub a: i128,
    pub b: i128,
    pub amount: i128,
}
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {

    // ...

    match token.mock_all_auths().try_transfer(&a, &b, &input.amount) {
        Ok(Ok(())) => {} // Expected success
        Err(Ok(_)) => {} // Expected error
        Ok(Err(_)) => panic!("success with wrong type returned")
        Err(Err(_)) => panic!("unrecognised error")
    }
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {

    // ...

    match token.mock_all_auths().try_transfer(&a, &b, &input.amount) {
        Ok(Ok(())) => {} // Expected success
        Err(Ok(_)) => {} // Expected error
        Ok(Err(_)) => panic!("success with wrong type returned")
        Err(Err(_)) => panic!("unrecognised error")
    }
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {

    // ...

    match token.mock_all_auths().try_transfer(&a, &b, &input.amount) {
        Ok(Ok(())) => {} // Expected success
        Err(Ok(_)) => {} // Expected error
        Ok(Err(_)) => panic!("success with wrong type returned")
        Err(Err(_)) => panic!("unrecognised error")
    }
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {

    // ...

    match token.mock_all_auths().try_transfer(&a, &b, &input.amount) {
        Ok(Ok(())) => {} // Expected success
        Err(Ok(_)) => {} // Expected error
        Ok(Err(_)) => panic!("success with wrong type returned")
        Err(Err(_)) => panic!("unrecognised error")
    }
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {

    // ...

    match token.mock_all_auths().try_transfer(&a, &b, &input.amount) {
        Ok(Ok(())) => {} // Expected success
        Err(Ok(_)) => {} // Expected error
        Ok(Err(_)) => panic!("success with wrong type returned")
        Err(Err(_)) => panic!("unrecognised error")
    }
});
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING AND PROPERTY TESTING

```rust
fuzz_target!(|input: Input| {

    // ...

    match token.mock_all_auths().try_transfer(&a, &b, &input.amount) {
        Ok(Ok(())) => {} // Expected success
        Err(Ok(_)) => {} // Expected error
        Ok(Err(_)) => panic!("success with wrong type returned")
        Err(Err(_)) => panic!("unrecognised error")
    }
});
```

INVARIANTS

INVARIANTS

```
assert!(all_balances ⩾ 0);
```

INVARIANTS

ASSERTING INVARIANTS

```rust
pub fn mint(env: &Env, to: Address, amount: i128) {
    // ...
    Self::assert_balances_gte_zero(env, &[to]);
}

pub fn transfer(env: &Env, from: Address, to: Address, amount: i128) {
    // ...
    Self::assert_balances_gte_zero(env, &[from, to]);
}

fn assert_balances_gte_zero(env: &Env, addrs: &[Address]) {
    for addr in addrs {
        assert!(Self::balance(env, addr.clone()) ≥ 0);
    }
}
```

ASSERTING INVARIANTS

```rust
pub fn mint(env: &Env, to: Address, amount: i128) {
    // ...
    Self::assert_balances_gte_zero(env, &[to]);
}

pub fn transfer(env: &Env, from: Address, to: Address, amount: i128) {
    // ...
    Self::assert_balances_gte_zero(env, &[from, to]);
}

fn assert_balances_gte_zero(env: &Env, addrs: &[Address]) {
    for addr in addrs {
        debug_assert!(Self::balance(env, addr.clone()) ≥ 0);
    }
}
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING INVARIANTS

```rust
#[derive(Arbitrary)]
pub enum Action {
    Mint(<Address as SorobanArbitrary>::Prototype, i128),
    Balance(<Address as SorobanArbitrary>::Prototype),
    Transfer(
        <Address as SorobanArbitrary>::Prototype,
        <Address as SorobanArbitrary>::Prototype,
        i128,
    ),
}

fuzz_target!(|actions: std::vec::Vec<Action>| {
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING INVARIANTS

```
fuzz_target!(|actions: std::vec::Vec<Action>| {
    let env = Env::default();
    env.mock_all_auths();
    let id = env.register(Token, (&Address::generate(&env),));
    let token = TokenClient::new(&env, &id);
    for a in actions {
        match a {
            Action::Mint(addr, amount) => _ = token.try_mint(&addr.into_
            Action::Balance(addr) => _ = token.try_balance(&addr.into_va
            Action::Transfer(from, to, amount) => _ = token.try_transfer
        }
    }
    assert_balances_gte_zero(env);
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING INVARIANTS

```
fuzz_target!(|actions: std::vec::Vec<Action>| {
    let env = Env::default();
    env.mock_all_auths();
    let id = env.register(Token, (&Address::generate(&env),));
    let token = TokenClient::new(&env, &id);
    for a in actions {
        match a {
            Action::Mint(addr, amount) => _ = token.try_mint(&addr.into_
            Action::Balance(addr) => _ = token.try_balance(&addr.into_va
            Action::Transfer(from, to, amount) => _ = token.try_transfer
        }
    }
    assert_balances_gte_zero(env);
```

```
$ cargo fuzz init
$ cargo +nightly fuzz run --sanitizer=thread fuzz_target_1
```

FUZZING INVARIANTS

```
fuzz_target!(|actions: std::vec::Vec<Action>| {
    let env = Env::default();
    env.mock_all_auths();
    let id = env.register(Token, (&Address::generate(&env),));
    let token = TokenClient::new(&env, &id);
    for a in actions {
        match a {
            Action::Mint(addr, amount) => _ = token.try_mint(&addr.into_
            Action::Balance(addr) => _ = token.try_balance(&addr.into_va
            Action::Transfer(from, to, amount) => _ = token.try_transfer
        }
    }
    assert_balances_gte_zero(env);
```

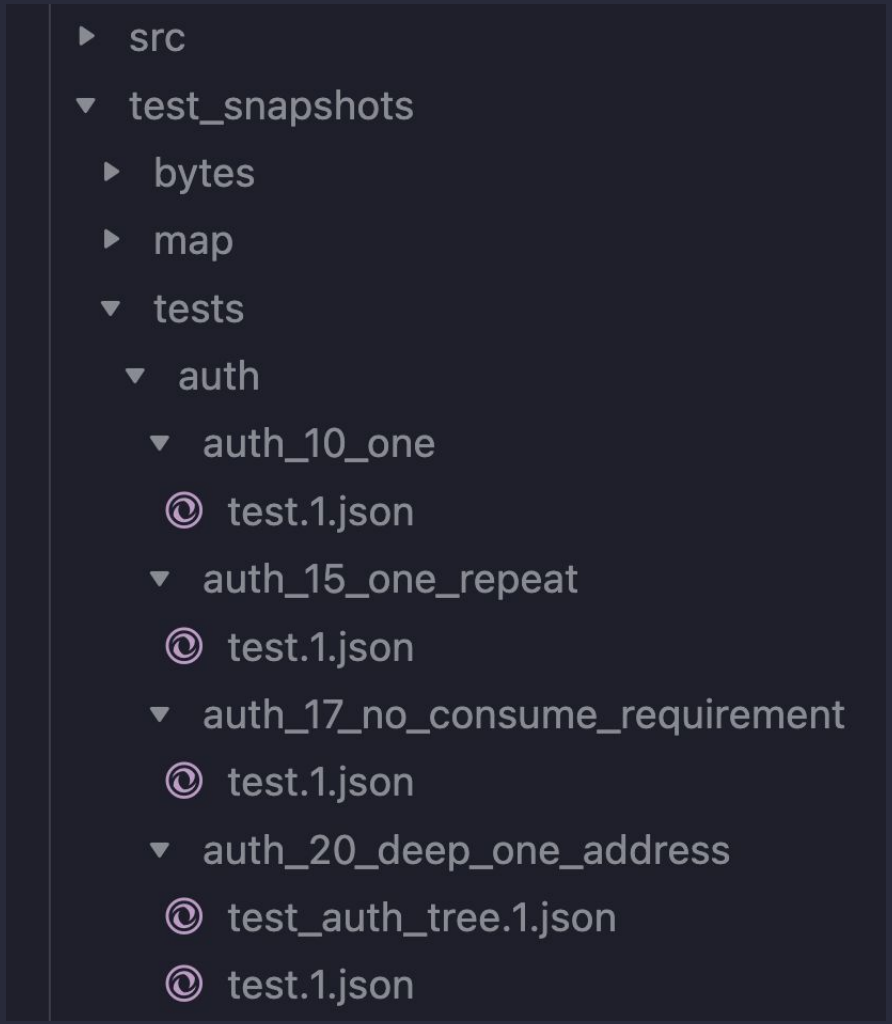
DIFFERENTIAL TESTING

DIFFERENTIAL TESTING

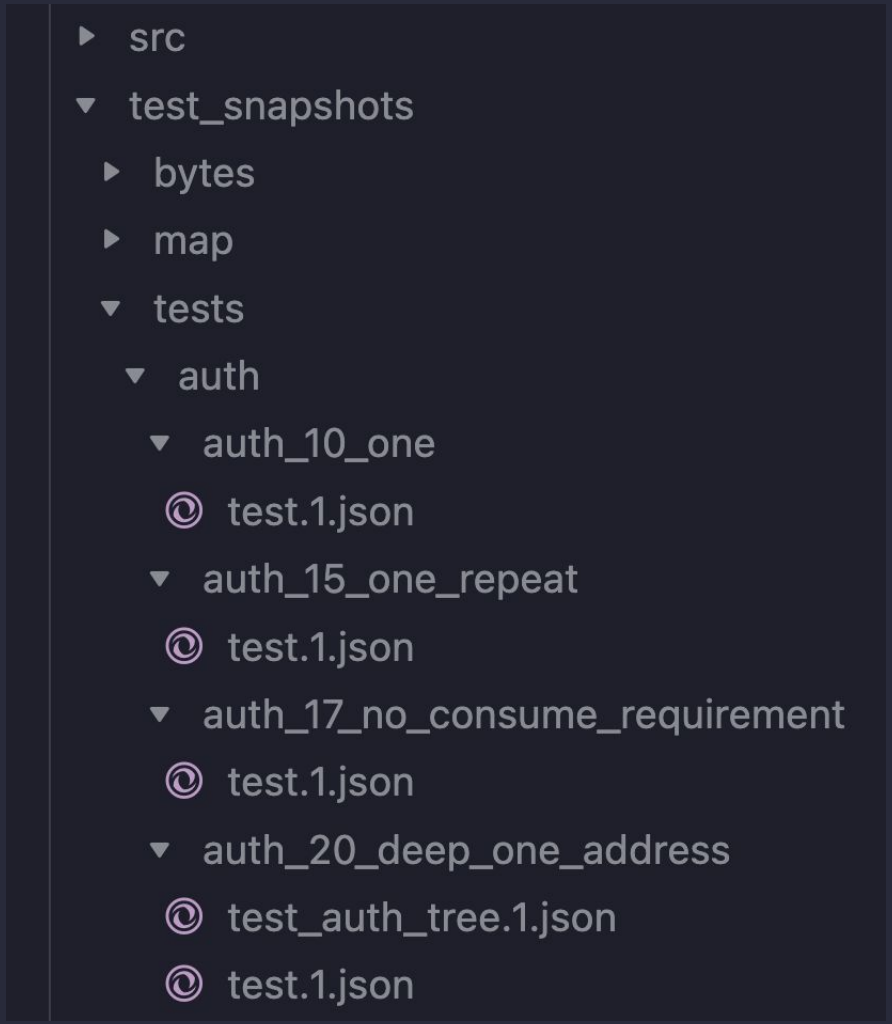
DIFFERENTIAL TESTING

DIFFERENTIAL TESTING

DIFFERENTIAL TESTING



DIFFERENTIAL TESTING



```
$ stellar contract fetch --id C...  --out-file token.wasm
```

DIFFERENTIAL TESTING

```
$ stellar contract fetch --id C...  --out-file pause.wasm
```

```rust
mod token {
    soroban_sdk::contractimport!(file = "token.wasm");
}
```

DIFFERENTIAL TESTING

DIFFERENTIAL TESTING

```rust
#[test]
fn test() {
    assert_eq!({
        let env = Env::default();
        let id = env.register(Token, (&Address::generate(&env),));
        let token = TokenClient::new(&env, &id);
        let a = Address::generate(&env);
        let b = Address::generate(&env);
        token.mock_all_auths().mint(&a, &10);
        token.mock_all_auths().transfer(&a, &b, &2);
        (token.balance(&a), token.balance(&b))
    }, {
        let env = Env::default();
        let id = env.register(pause::WASM, (&Address::generate(&env),));
        let token = TokenClient::new(&env, &id);
        let a = Address::generate(&env);
        let b = Address::generate(&env);
        token.mock_all_auths().mint(&a, &10);
        token.mock_all_auths().transfer(&a, &b, &2);
        (token.balance(&a), token.balance(&b))
    });
}
```

1. TESTING WITH REAL DEPENDENCIES
2. TESTING WITH REAL DATA
3. FUZZING AND PROPERTY TESTING
4. INVARIANTS
5. DIFFERENTIAL TESTING

1. TESTING WITH REAL DEPENDENCIES
2. TESTING WITH REAL DATA
3. FUZZING AND PROPERTY TESTING
4. INVARIANTS
5. DIFFERENTIAL TESTING

1. TESTING WITH REAL DEPENDENCIES
2. TESTING WITH REAL DATA
3. FUZZING AND PROPERTY TESTING
4. INVARIANTS
5. DIFFERENTIAL TESTING
6. FORMAL VERIFICATION

1. TESTING WITH REAL DEPENDENCIES
2. TESTING WITH REAL DATA
3. FUZZING AND PROPERTY TESTING
4. INVARIANTS
5. DIFFERENTIAL TESTING
6. FORMAL VERIFICATION

“

THE EARLIER A BUG IS FOUND, THE CHEAPER IT IS TO FIX.

—Karen N. Johnson

“

THE EARLIER A BUG IS FOUND, THE CHEAPER IT IS TO FIX.

—Karen N. Johnson

Stellar

TEST!

 ___leigh___

 ___leigh___

 leighmcculloch