Foundry 101 Cheat Sheet

1. Basics & Installation

Tool	Command	Description
Vanilla Foundry	foundryup	Installs or updates Foundry.
zkSync Foundry	foundryup-zksync	Installs Foundry for zkSync development.

2. Core Commands

Command	Description
forge init	Initializes a new Foundry project.
anvil	Starts a local Ethereum node for development.
cast to-base	Converts data to base representation.
forge build	Compiles the smart contracts.
forge script	Runs scripts written for deployment/testing.
forge test -vv	Runs tests with verbose output.
forge coverage	Displays code coverage of tests.
cast call $CONTRACT_ADDRESS --rpc-url $RPC_URL	Calls a view function on a deployed contract.
cast send $CONTRACT_ADDRESS --rpc-url $RPC_URL --private-key $PRIVATE_KEY	Calls a state-changing function.

3. Testing Strategies

Type	Description	Command Example
Unit	Tests specific code sections.	forge test -vvv --fork-url $SEPOLIA_RPC_URL
Integration	Tests how different parts of the code interact.	
Forked	Simulates real-world blockchain environments.	forge coverage --fork-url $SEPOLIA_RPC_URL

Staging	Tests on a real environment (testnet/mainnet).	

4. Mock Contracts

Use Case	Description
Deploy Mocks	Deploys mock contracts on local Anvil chains.
Track Contract Addresses	Tracks addresses across different environments.
File Location	script/helperconfig.sol

5. Transactions

State	Description
Before vm.startBroadcast()	Transactions are simulated.
After vm.startBroadcast()	Transactions are real and sent to the blockchain.

6. VM Cheat Sheet

Command	Description
vm.expectRevert();	Ensures the next transaction reverts, otherwise the test fails.
vm.prank(USER);	Mocks the next transaction as being sent by USER.
address USER = makeAddr("user");	Creates a dummy user without ETH.
vm.deal(USER, STARTING_BALANCE);	Gives USER some ETH for testing.
hoax(ADDRESS, SEND_VALUE);	Combines prank and deal: creates a user and assigns ETH.
vm.startPrank(USER);	Starts a persistent impersonation session for USER.
vm.stopPrank();	Ends the persistent impersonation session.

7. Best Practices

- Storage Variables: Prefix with : s_variableName
- Function Parameters: Prefix with : _paramName

- Immutable variables: Prefix with : i_paramName

8. Gas Optimization and Error Handling

A. Custom Errors vs. Strings in require

Using strings in require statements is less gas efficient. Instead, use custom errors:

- Inefficient:

```
require(msg.value >= i_entranceFee, "Not enough ETH sent");
```

- More Efficient (Using if and revert):

```
error NotEnoughEth();
if (msg.value < i_entranceFee) {
        revert NotEnoughEth();
}
```

- Most Efficient (Custom Errors with require):

```
error NotEnoughEth();
require(msg.value >= i_entranceFee, NotEnoughEth());
```

Note: Custom errors are available in the latest Solidity versions.
Best Practice: Name your errors with the contract name as a prefix (e.g., Raffle_NotEnoughEth()).

B. Storage vs. Memory

Operation	Gas Cost
Storage (sload/sstore)	100 Gas
Memory (mload/mstore)	3 Gas

9. Gas & Storage Insights

Command	Description
forge snapshot	Generates a file storing gas consumption per test.
forge inspect CONTRACT_NAME storageLayout	Displays variable storage layout in a contract.
cast storage CONTRACT_ADDRESS INDEX	Shows the value of a variable at a storage index.

10. Constants & Immutables

- Constants and immutables are not stored in storage
- They are part of the contract's bytecode.

11. Foundry Configuration

Config Line	Description
ffi=1	Allows Foundry to run commands on the machine. (but it is dangerous, enable when needed only)

12. Function Signatures & Call Data

What is a function selector: Function selectors are unique identifiers for smart contract functions, used by the EVM to understand which function is being called during a transaction.

Command	Description
cast sig "FUNCTION_NAME()"	Returns the function selector.
cast --calldata-decode "FUNCTION_NAME()" FUNCTION_SIGNATURE	Decodes call data for transactions with parameters.

13. Project & Contract Organization

Organizing your Solidity contracts enhances readability and maintainability. Follow this recommended order:

1. Pragma Statements: Specify the compiler version.
2. Import Statements: Include dependencies.
3. Interfaces: Define contract interfaces.
4. Libraries: Implement reusable code.
5. Contracts: Define contract logic.

Within each contract, library, or interface, order elements as follows:

- Type Declarations: Custom types or structs.
- State Variables: Variables storing contract state.
- Events: Log activities or changes.
- Modifiers: Restrict or alter function behavior.
- Functions: Executable code blocks.

Inside each function, maintain this order:

- Constructor: Initializes the contract.
- Receive Function: Handles plain Ether transfers.

- Fallback Function: Handles non-existent function calls or plain Ether transfers.
- External Functions: Callable from external contracts or accounts.
- Public Functions: Callable internally and externally.
- Internal Functions: Callable only within the contract or derived contracts.
- Private Functions: Callable only within the contract.

14. Events in Solidity

A. Why Use Events?
   - Migration: Simplifies tracking contract state changes during upgrades.
   - Frontend Indexing: Events allow off-chain applications to index and query changes easily.

B. Indexed vs. Non-Indexed Parameters
   - Indexed Parameters:
     - Cost more gas (as they are stored in the log topics, which are searchable).
     - Make it easier to search and filter events.
   - Non-Indexed Parameters:
     - Encoded in the event data (cheaper but not directly searchable).

C. Best Practice
   - Emit an Event Whenever You Update Storage: This ensures that any state change is traceable off-chain.

15. Global Variables & Conventions

- `block.timestamp`: A globally available unit for the current block's timestamp.
- Constants: Use CAPITAL LETTERS (e.g., `uint256 constant MAX_SUPPLY = 1000;`).
- Naming Conventions:
  - Storage Variables: Prefix with `s_` (e.g., `s_players`).
  - Function Parameters: Prefix with `i_` (e.g., `i_fee`).

16. Inheritance & Constructors

When a contract inherits from another, pass required parameters to the parent constructor in your child contract's constructor.

17. Enums & Structs

- Enums: Provide a way to create user-defined types with a finite set of values.

```
enum Status { Pending, Active, Inactive }
```

- Structs: Allow you to group variables (e.g., the Map struct in your Vault contract).

```
struct Player {
        address wallet;
        uint256 score;
}
```

18. Resetting Arrays

To reset an array in Solidity:

```
s_players = new address payable[](0);
```

19. CEI Pattern (Checks-Effects-Interactions)

Purpose: Prevents reentrancy by ensuring that:

- Checks: Validate conditions using require statements.
- Effects: Update the internal state.
- Interactions: Make external calls.

Reentrancy: Occurs when an external call re-enters the calling function before state changes are finalized. The CEI pattern helps mitigate this risk.

20. Chainlink Automations (Keepers) & VRF

A. Chainlink Automations:
   - checkUpkeep: Listens and determines if the function should be executed.
   - performUpkeep: Executes the function once conditions are met.

B. Chainlink VRF:
   - Used for generating random numbers in a verifiable and tamper-proof way.

21. Memory vs. Calldata vs. Storage

- Gas Efficiency Ranking:
  calldata > memory > storage
- Explanation:
  - Calldata: Read-only, external function arguments; cheapest.
  - Memory: Temporary, in-function variables; moderately expensive.
  - Storage: Persists on-chain; most expensive.

22. VM Cheat Sheet (Foundry Testing)

VM Command	Description
vm.warp(newTimestamp)	Sets the block timestamp to newTimestamp (useful for simulating time passage in tests).
vm.roll(newBlockNumber)	Sets the block number to newBlockNumber.
vm.recordLogs()	Begins recording emitted events during a transaction.
vm.getRecordedLogs()	Retrieves an array of recorded logs (Vm.Log[] memory entries) from the previous vm.recordLogs() session.
vm.expectRevert()	Expects the following transaction to revert; if not, the test fails.
vm.prank(USER)	Temporarily sets msg.sender to USER for the next transaction.
vm.deal(USER, STARTING_BALANCE)	Sets a balance for USER so they can send transactions in tests.
hoax(ADDRESS, SEND_VALUE)	Combines prank and deal: creates a dummy address with an initial balance and uses it as the sender for the next transaction.
vm.startPrank(USER); vm.stopPrank();	Begins and ends a persistent impersonation of USER for multiple transactions.

23. Fuzz Testing

- Purpose: Automatically tests your contracts with random inputs.
- Configuration in foundry.toml:

```
[fuzz]
runs = 1000  # Number of fuzz test runs
```

- Types of Fuzzing:
  - Stateless Fuzzing: Each test run is independent.
  - Stateful Fuzzing: Test runs that depend on the previous state.
  - Formal Verification: Proving properties of your contract mathematically.

24. Type Conversions & Address Casting

- Example Conversion:

```
address addr = address(uint160(i));
```

This converts a number i into an Ethereum address.

25. Command Obfuscation in Makefiles

When running commands in Makefiles (e.g., for deployment), sensitive information (like passwords) can be obfuscated or hidden from terminal output for security.

26. Advanced Topics

A. Pivoting, Lateral Movement & Evasive Testing

- Pivoting: Using a compromised system to attack other systems in the network.
- Lateral Movement: Moving from one compromised system to another to gain broader access.
- Evasive Testing: Techniques to bypass security defenses during penetration testing.

27. Commands Reference Table

Command	Usage & Description
foundryup	Installs/updates vanilla Foundry.
foundryup-zksync	Installs Foundry for zkSync.
forge init	Initializes a new Foundry project.
anvil	Starts a local Ethereum node.
forge build	Compiles smart contracts.
forge script	Runs deployment or testing scripts.
forge test -vv	Runs tests with verbose logging.
forge coverage	Generates a code coverage report for tests.
cast call <CONTRACT> --rpc-url <RPC_URL>	Calls a view function on a deployed contract.
cast send <CONTRACT> --rpc-url <RPC_URL> --private-key <KEY>	Sends a transaction for a state-changing function.
forge snapshot	Generates a snapshot file containing gas consumption for each test.
forge inspect <CONTRACT> storageLayout	Shows the storage layout of a contract.

cast storage <CONTRACT_ADDRESS> <INDEX>	Retrieves the value stored at a specific storage index.
cast sig "FUNCTION_NAME()"	Returns the function selector for a function signature.
cast --calldata-decode "FUNCTION_NAME()" <SIGNATURE>	Decodes the calldata of a transaction with parameters.