**AccessControl**

We assign role as Bytes32

e.g

bytes32 public constant MY\_ROLE = keccak256("MY\_ROLE");

This bytes32 hash will be the role and we will assign it to different address(accounts).

To restrict access to a function call, use {hasRole}:

function foo() public {

require(hasRole(MY\_ROLE, msg.sender))

…..

}

**In the constructor**

We granted (bytes32 hash ADMIN) as an admin and also

Address .

We also set Role as ADMIN.

This is necessary because we have modifier onlyRole() applied on some functions.

**grantRole**

If you want to grant role of ADMIN to some other address ,

Then put

Role address (BYTES32 ADMIN )

ADDRESS (That you want to give admin role.)

Note: we can assign same role to multiple address.

* **renounceRole**

Revokes `role` from the calling account.

|  |
| --- |
|  |
|  | \* |
|  | \* Roles are often managed via {grantRole} and {revokeRole}: this function's |
|  | \* purpose is to provide a mechanism for accounts to lose their privileges |
|  | \* if they are compromised (such as when a trusted device is misplaced). |
|  | \* |
|  | \* If the calling account had been revoked `role`, emits a {RoleRevoked} |
|  | \* event. |
|  | \* |
|  | \* Requirements: |
|  | \* |
|  | \* - the caller must be `account`. |

**hasRole(bytes32 role, address account)**

Returns `true` if `account` has been granted `role`.