Contents

[Cucumber – BDD [Behavior Driven Development] 1](#_Toc514001891)

[Parameter Handling 4](#_Toc514001892)

[Scenarios Data Table 6](#_Toc514001893)

[Scenarios Outline Data Table 8](#_Toc514001894)

[Scenario Hooks 9](#_Toc514001895)

[Background Steps 12](#_Toc514001896)

[Trick in Cucumber BDD 12](#_Toc514001897)

[Angular JS 12](#_Toc514001898)

[Directives 13](#_Toc514001899)

[Scope 14](#_Toc514001900)

[Angular Controllers 15](#_Toc514001901)

[Angular Modules 16](#_Toc514001902)

[Dependency Injection and Scope 16](#_Toc514001903)

[ng-click 17](#_Toc514001904)

[ng-src 18](#_Toc514001905)

[ng-model - Two way data binding 18](#_Toc514001906)

[ng-show and ng-hide 18](#_Toc514001907)

[ng-repeat – Looping 19](#_Toc514001908)

[Angular Services 20](#_Toc514001909)

[Angular with REST Services 24](#_Toc514001910)

[$http Service 27](#_Toc514001911)

[Log Service 29](#_Toc514001912)

[Formatting using Angular 29](#_Toc514001913)

[Sorting 32](#_Toc514001914)

[ng-include 34](#_Toc514001915)

[Routing 34](#_Toc514001916)

[Project Structure 36](#_Toc514001917)

[Jenkins 38](#_Toc514001918)

[Jenkins Setup 38](#_Toc514001919)

[How to use CLI(Command Line Interface in Jenkins) 42](#_Toc514001920)

[Jenkins: Basic Configuration 42](#_Toc514001921)

[Jenkins Jobs 43](#_Toc514001922)

[GIT 46](#_Toc514001923)

# Angular JS

|  |  |
| --- | --- |
| Advantage   * Dependency Injection * Two ways data binding 🡪 The change in Model layer changes the view layer automatically * Easy to develop MVC Application |  |

Using Angular Expression

1. {{10+10}} 🡪20
2. {{1==1} } 🡪true
3. If it’s a plain old Javascript object : {{{name:'Amit',age:'30'}.name}} 🡪 Amit
4. If its an Array : {{['AA','BB','CC'][2]}} 🡪CC

### Scope

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| <body ng-init="**quantity=5**">  <div id="td-wrapper" ng-controller="demoController">  <div id="td-container">  **<p ng-if="quantity>2">Quantity is Greater than 2</p>**  </div>  </div>  </body> | | Here the quantity variable is created in ng-init directive but we are using it in ng-if directive. So now the question is how the variable talks to each other if they are created in one directive and used in another directive.  **Answer**  When ever a variable is created in directive they are created in common space call “**scope**”. So all the directive shares a common space, and this is a way they share variable | | |
| **quantity = 5** means we are creating a property(quantity) in scope object and assigning a value “5” to it.  **ng-if="quantity>2"** means we are reading a property from the scope object.  Refer the fig below for reference. | |  | | |
| ng-bind  * Now suppose we want to display the variable created in scope we can use **ng-bind** directive * Expression can also be used to display | | <body ng-init="quantity=5">  <div id="td-wrapper" ng-controller="demoController"> <div id="td-container">  <p>Quantity is <span ng-bind="quantity"></span></p>  <p>using Expression{{quantity}}</p>  </div>  </div>  </body> |

### Angular Controllers

Now suppose we have to show the data in view layer after some processing a business logic so to process this business logic we create a controller.

* controller="**demoController 🡪** says ,display the DOM inside controller only after procession the controller function
* Angular says we have to register a function to the controller [demoAppModule.controller("demoController",**demoFunction**);]

|  |  |
| --- | --- |
| HTML  <html ng-app="demoApp">  <head>  <link rel="stylesheet" href="css/default.css">  <script src="js/angular.min.js"></script>  <script src="js/init.js"></script>  </head>  <body ng-init="quantity=5">  <div id="td-wrapper" ng-controller="demoController">  <div id="td-container">  <p>Quantity is <span ng-bind="quantity"></span></p>  <p>Using Expression{{quantity}}</p>  </div>  </div> </body></html> | **JS**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("demoController",demoFunction);  function demoFunction(){  console.log("Hello");  }  **OR**  var demoAppModule = angular.module("demoApp ",[]);  var demoController = function($scope){  $scope.pageHeading="Angular Tutorial";  }  demoAppModule.controller("**demoController** ", demoController);   * **Angular is a global variable** |
| **ATTACHING A COMPLEX OBJECT**  **HTML**  Employee Name : {{employee.firstName}}  Employee LName : {{employee. lastName}}  Employee Gender : {{employee. gender }} | **JS**  var employeeMgmtApp = angular.module("employeeMgmtApp",[]);  var employeeObj ={  firstName :'Amit',  lastName :'Sinha',  gender:'Male'  }  var employerController = function($scope){  $scope.pageHeading="Angular Tutorials";  $scope.employee = employeeObj;  }  employeeMgmtApp.controller("employeeController", employerController); |

### Angular Modules

* This defines the angular application

Syntax 🡪 var demoAppModule = angular.module("demoApp",[]);

* The Angular Module a phone book, where we register the contact information. Similary we register the angular element (like controllers) to use it.
* It’s a global place for creating ,registering and retrieving angular modules
* A module is a collection of directives, contollers and other elements.
* Refer above for referece

The second argument is used to import other modules

|  |  |
| --- | --- |
| **Step 1 : Create a standalone module that will be imported**  var myModule = angular.module("**moduleApp**",[]);  myModule.controller("myController",myController);  function myController($scope){  $scope.myMessage ="This message is from my Controller";  } | **Step 2 : import the standalone module in your module**  var demoAppModule = angular.module("demoApp",["**moduleApp**"]); |
| **HTML**  <html ng-app="demoApp">  <body >  <div ng-controller="myController">  **<p>{{myMessage}}</p>**  </div>  </html> | * The myContoller is used as if it is defined in demoApp module * Multiple modules can imported in a module (comma seperated). |

### Dependency Injection and Scope

We usually access the scope variable in the view layer .So after business process the controller has to set the final output to scope variable to that view layer can access it. The scope variable can be accessed in controller bu injecting it in the controller itself.

|  |  |
| --- | --- |
| **HTML**  <p>Time is **{{timeValue}}**</p> | **JS**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("demoController",showTime);  function showTime**($scope)**{  var d = new Date();  **$scope.timeValue = d.toTimeString();**  } |

* The function showTime has been injected with scope object and the final result is stored in scope [timeValue].

### ng-click

* The ng-click=” updateTime()” means we are creating a function in scope
* So in JS we have to assign a function as $scope.updateTIme = function () { }

|  |  |
| --- | --- |
| **HTML**  <div id="td-container">  <p>Time is {{timeValue}}</p>  <p><a class="td-button td-button-primary td-button-compact" ng-click="updateTime()"><span class="td-button-label">Update Time</span></a></p>  </div> | **JS**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("demoController",showTime);  function showTime($scope){  $scope.updateTime= function(){  var d = new Date();  $scope.timeValue = d.toTimeString(); } } |
| **EXAMPLE 2**  <table class="td-table td-table-stripe-column td-table-border-row td-table-border-column" width="100%">  <tbody> <tr class="td-table-row-first">  <th class="td-table-heading-top">S.no.</th>  <th class="td-table-heading-top">Technology Name</th>  <th class="td-table-heading-top">Relevant Exp </th>  <th class="td-table-heading-top">Edit Exp </th>  </tr>  <tr ng-repeat="emp in employee.techworkedOn">  <td class="td-copy-sub">{{$index+1}}</td>  <td class="td-copy-sub">{{emp.tech}}</td>  <td class="td-copy-sub">{{emp.exp}}</td>  <td class="td-copy-sub"><a href="#" ng-click="setExperience(emp)" class="td-button td-button-primary"><span class="td-button-label">+</span></a></td> </tr> </tbody> </table> | var employeeMgmtApp = angular.module("employeeMgmtApp",[]);  var employeeObj ={  techworkedOn:[  {tech: "Java",exp:3},  {tech: "CQ",exp:2},  {tech: "Spring",exp:1}  ]  }  var employerController = function($scope){  $scope.employee = employeeObj;  $scope.setExperience = function(employee){  employee.exp++;  }  }  employeeMgmtApp.controller("employeeController", employerController); |

### ng-src

|  |  |
| --- | --- |
| <img ng-src="{{employee.profileimg}}"/> | var employeeMgmtApp = angular.module("employeeMgmtApp",[]);  var employeeObj ={  firstName :'Amit',  lastName :'Sinha',  gender:'Male',  profileimg:'../images/employee.jpg'  }  var employerController = function($scope){  $scope.employee = employeeObj;  }  employeeMgmtApp.controller("employeeController", employerController); |

### ng-model - Two way data binding

VIEW LAYER CONTROLLER

* ng-model model can be used for two ways data binding. For example user enters a data in the view 🡪updates the scope variable in controller 🡪 view layer retrieve the scope variable

|  |  |
| --- | --- |
| <p><input type="text" ng-model="userName"/></p>  <p ng-bind="userName"></p> | var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("demoController",showTime);  function showTime($scope){  $scope.userName="";  } |

* The value ng-model in textfield should be same as scope variable and same can be retrieve back in view

### ng-show and ng-hide

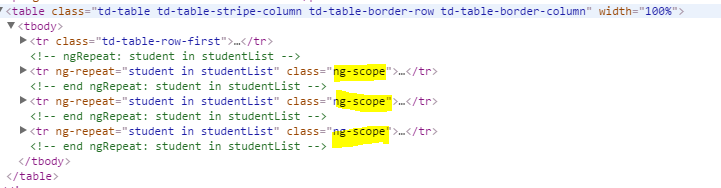
|  |
| --- |
| <p><input type="checkbox" ng-model="**showContent**"/></p>  <p ng-show="showContent">Some when checked </p>  <p ng-hide="showContent">Show when unchecked</p> |
|  |
| * ng-show and ng-hide adds class ng-hide to hide the DOM element [adds display:none!important in css] * but ng-if completely removes the HTML element from DOM tree |

### ng-repeat – Looping

|  |
| --- |
|  |
| **HTML**  <table class="td-table td-table-stripe-column td-table-border-row td-table-border-column" width="100%">  <tbody>  <tr class="td-table-row-first">  <th class="td-table-heading-top" >S.No.</th>  <th class="td-table-heading-top">First Name</th>  <th class="td-table-heading-top">Last Name</th>  </tr>  <tr ng-repeat="student in studentList">  <td class="td-copy-sub">{{$index+1}}</td>  <td class="td-copy-sub">{{student.firstName}}</td>  <td class="td-copy-sub">{{student.lastName}}</td>  </tr>  </tbody>  </table> |
| **JS**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("demoController",showTime);  function showTime($scope){  $scope.studentList=[  {"firstName": "Student 1 First Name","lastName": "Student 1 Last Name"},  {"firstName": "Student 2 First Name","lastName": "Student 2 Last Name"},  {"firstName": "Student 3 First Name","lastName": "Student 3 Last Name"}  ];  } |

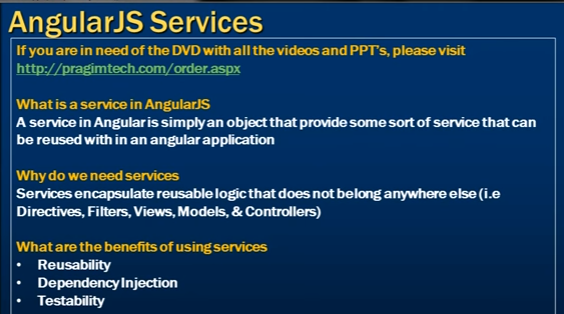
Important points for ng-repeat

* The ng-repeat create scope object for every iteration



Reason? 🡪

## Angular Services



|  |  |
| --- | --- |
| * Services are reusable elements of Angular code * Dependency Injection : The Services can be injected in any controller or any other service. * This used to share data across the controllers. * This shared data can be   + *Values*   + *Objects*   + *Functions* | Some important Angular out of Box services   * *Value Service* * *Constant Service* * *Service Service* * *Factory Service* * *Http Service* |

#### Value service

The main intention of creating services is to share data across controllers.

**Value Service** 🡪 To understand this service, let’s consider an example- Suppose we have a website in which we need to show same content in both header and footer. So we use the value service when we want to share a value across multiple controllers.

**Without Using Service**

|  |  |
| --- | --- |
| **HTML**  For Header  <div class="td-layout-row" ng-controller="headerController">  <h1 class="td-copy-align-centre" ng-bind="appTitle"></h1>  </div>  For Footer  <div class="td-layout-row" ng-controller="footerController">  <p class="td-copy-legal td-copy-align-centre" ng-bind="appTitle"></p>  </div> | * For each controller we have to create separate a scope variable(appTitle) * The scope variable of header Contoller can’t be user in Footer Controller as reg scope variable are visible to respective controller * Now the objective of service is to share the data across the contollers and solve the above problem * The value service I used to share a “**VALUE**” across the controller |
| **JS**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("headerController",headerMessage);  demoAppModule.controller("footerController",footerMessage);  function headerMessage($scope){  **$scope.appTitle="TD Bank";**  }  function footerMessage($scope){  **$scope.appTitle="TD Bank";**  } |

**Solution: Share the VALUE using Value Service**

|  |  |  |
| --- | --- | --- |
| **HTML**  For Header  <div class="td-layout-row" ng-controller="headerController">  <h1 class="td-copy-align-centre" ng-bind="appTitle"></h1>  </div>  For Footer  <div class="td-layout-row" ng-controller="footerController">  <p class="td-copy-legal td-copy-align-centre" ng-bind="appTitle"></p>  </div> | | * The value service used t o share a value across the controller * The value need not to be just a String value. It can a Object or function as well |
| **IF THE VALUE IS A JAVASCRIPT OBJECT**  demoAppModule.value("valueSvc",  {  “appTitle” : “TD Bank”,  “author” : “Mr Foo”,  “company”: “ABC Ltd.”  });  function headerMessage(valueSvc,$scope){  $scope.appTitle=valueSvc.appTitle;  }  function footerMessage(valueSvc,$scope){  $scope.appTitle=valueSvc.appTitle;  }  \*\*This is an example to show how service can be injected in controller. |
| **JS**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("headerController",headerMessage);  demoAppModule.controller("footerController",footerMessage);  **demoAppModule.value("valueSvc", "TD Bank");**  function headerMessage(valueSvc,$scope){  $scope.appTitle=valueSvc;  }  function footerMessage(valueSvc,$scope){  $scope.appTitle=valueSvc;  } | |
| **IF THE VALUE IS A FUNCTION**  var demoAppModule = angular.module("demoApp",[]);  demoAppModule.controller("headerController",headerMessage);  demoAppModule.controller("footerController",footerMessage);  demoAppModule.value("valueSvc", {  "appTitle" : "TD Bank",  "company" : "ABC Ltd.",  "author" :"Mr.Foo"  });  demoAppModule.value("LoggingSvc", function(){  console.log("Logging Data");  });  function headerMessage(valueSvc,$scope,**LoggingSvc**){  $scope.appTitle=valueSvc.appTitle;  LoggingSvc();  }  function footerMessage(valueSvc,$scope,**LoggingSvc**){  $scope.appTitle=valueSvc.appTitle;  LoggingSvc();  }  \*\*This is an example to show how service can be injected in controller. | Constant Service This service can be used when we know the value is not going to change. This can be declared same as VALUE service and all that which applies to VALUE service holds good for both  Having one Diffrence? | |

#### Factory Service

In value or constant service we usually pass the value or function inline. For example,

|  |  |
| --- | --- |
| demoAppModule.value("valueSvc", {  "appTitle" : "TD Bank",  "company" : "ABC Ltd.",  "author" :"Mr.Foo"  }); | demoAppModule.value("valueSvc", “Hello World”); |
| demoAppModule.value("LoggingSvc", function(){  console.log("Logging Data");  }); |

What if we need to keep the function standalone rather than calling it inline (as in value and constant service). To resolve this problem we have a Factory Service. We can have a function to prepare the output and return the output and finally assign it to service.

|  |  |  |
| --- | --- | --- |
| demoAppModule.factory("AppDataFactorySvc",function(){  return "Hello from factory";  });  function headerMessage(valueSvc,$scope, AppDataFactorySvc){  $scope.appTitle=valueSvc.appTitle;  console.log(AppDataFactorySvc);  } | | * Here the function is returning a value which id finally getting assigned to AppDataFactorySvc * It is the return value which is assigned to AppDataFactorySvc not the function it self as in value service * That why syntax is console.log(AppDataFactorySvc); not console.log(AppDataFactorySvc**()**); |
| **IF ITS AN OBJECT**  demoAppModule.factory("AppDataFactorySvc",function(){  **return {**  **"appTitle" : "TD Bank",**  **"company" : "ABC Ltd.",**  **"author" :"Mr.Foo"**  **};**  });  function headerMessage(valueSvc,$scope, AppDataFactorySvc){  $scope.appTitle=valueSvc.appTitle;  console.log(**AppDataFactorySvc.appTitle**);  } | | |
| **IF IT’S A STANDALONE FUNCTION**  function **prepareData**(){  var value={};  value.appTitle = "TD Bank ltd.";  value.company = "ABC Ltd.";  value.author = "Mr.Foo";  return value;  } | demoAppModule.factory("AppDataFactorySvc",**prepareData**);  function headerMessage(valueSvc,$scope, **AppDataFactorySvc**){  $scope.appTitle=valueSvc.appTitle;  **console.log(AppDataFactorySvc.appTitle);**  } | |
| **DEPENDENCY INJECTION OF SERVICE INSIDE ANOTHER SERVICE**  demoAppModule.value("**AppNameSev**","TD Bank Ltd.");  function prepareData(**AppNameSev**){  var value={};  **value.appTitle = AppNameSev;**  value.company = "ABC Ltd.";  value.author = "Mr.Foo";  return value;  }  demoAppModule.factory("AppDataFactorySvc",**prepareData**);  function headerMessage(valueSvc,$scope, **AppDataFactorySvc**){  $scope.appTitle=valueSvc.appTitle;  **console.log(AppDataFactorySvc.appTitle);**  } | | |

## Angular with REST Services

Angular has provided some out of box sevice to make REST API call(Any HTTP call like GET/POST/PUT etc..) to REST Endpoint. **That is $http Service** .

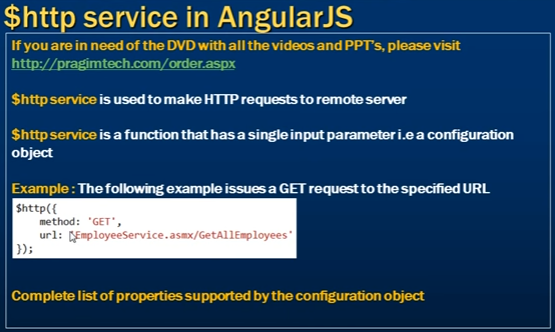
For the training pupose we need to create a REST API so that $http service can make REST API call to that service. So for training sake we are faking the the REST API. This can be achived by using **JSON SERVER.**

|  |  |
| --- | --- |
| **Steps to create a FAKE REST API**   1. Create a JSON file(data.json)[ <https://api.randomuser.me/?results=10> can be used to create a JSON of profiles] in the same folder where your HTML pages are lying(preferably) 2. Write the JSON – This will behave as response of REST API call using $http angular service 3. Navigate to the folder where the JSON file is present 4. And start the JSON SERVER - **json-server --port 4000 data.json**   After starting the JSON server. It can be verified in the browser by making the REST call directly - <http://localhost:4000/contacts/2> - as shown below. |  |

**Sample data.json file**

|  |
| --- |
| {  "contacts": [  {  **"id": 1,**  "gender": "male",  "name": {  "title": "mr",  "first": "romain",  "last": "hoogmoed"  },  "picture": {  "large": "https://randomuser.me/api/portraits/men/83.jpg",  "medium": "https://randomuser.me/api/portraits/med/men/83.jpg",  "thumbnail": "https://randomuser.me/api/portraits/thumb/men/8.jpg"  }  },  {  **"id": 2,**  "gender": "male",  "name": {  "title": "mr",  "first": "Rob",  "last": "Stark"  },  "picture": {  "large": "https://randomuser.me/api/portraits/men/84.jpg",  "medium": "https://randomuser.me/api/portraits/med/men/84.jpg",  "thumbnail": "https://randomuser.me/api/portraits/thumb/men/84.jpg"  }  },  {  "**id": 3,**  "gender": "female",  "name": {  "title": "Miss",  "first": "Emli",  "last": "Watson"  },  "picture": {  "large": "https://randomuser.me/api/portraits/women/84.jpg",  "medium": "https://randomuser.me/api/portraits/women/women/84.jpg",  "thumbnail": "https://randomuser.me/api/portraits/thumb/women/84.jpg"  }  }  ]  } |

### $http Service

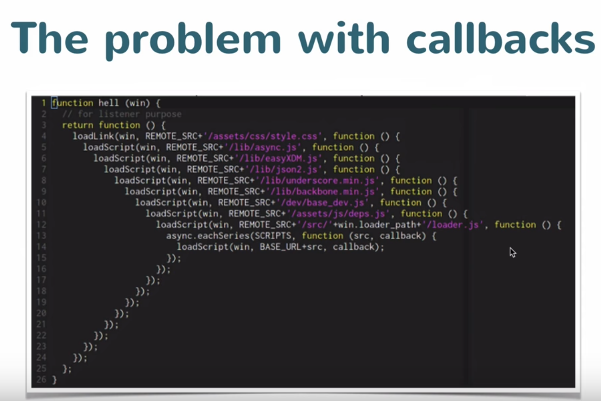


|  |
| --- |
| **HTML**  <div class="td-layout-column td-layout-grid6 td-layout-column-last">  <div class="td-callout td-callout-primary td-cs-primary" ng-repeat="profile in profileData">  <div class="td-callout-content">  <div class="td-layout-row">  <div class="td-layout-column td-layout-grid2 td-layout-column-first">  <img ng-src="{{profile.picture.thumbnail}}" alt="{{profile.picture.thumbnail}}"/>  </div>  <div class="td-layout-column td-layout-grid4 td-layout-column-first">  <p class="td-copy-sub">Name : {{profile.name.title}} {{profile.name.first}} {{profile.name.last}}</p>  <p class="td-copy-sub">Gender : {{profile.gender}}</p>  </div>  </div>  </div>  </div>  </div> |
| **JS**  $http.get('http://localhost:4000/contacts').then(function(response){  $scope.profileData = response.data;  });  Here we are using a get method a of http service |
| We can use the http as function which has one single parameter i.e configuration object. For Example  var employeeController = function($scope**,$http,$log**){  $http({  method:'GET',  url :'http://localhost:4000/contacts',  }).then(function(response){  $scope.profileData = response.data;  $log.info($scope.profileData);  **},function(reason){**  **$log.info(reason);**  **}**); |
| * Here the function in the then method is the success callback method. On the similar way we can have error callback function as well * Even we can keep the success and error callback method standalone and refer in http function. |
| **KEEP SUCCESS AND ERROR CALLBACK FUNCTION STANDALONE**  var successCallbackFunction = function(response){  $scope.profileData = response.data;  $log.info($scope.profileData);  }  var errorCallbackFunction = function(reason){  $log.info(reason);  }  $http({  method:'GET',  url :'http://localhost:4000/contacts',  }).then(successCallbackFunction,errorCallbackFunction); |
| IMPORTANT |

In Asych Javascript programming, we say the browser not to get blocked till we get the response

|  |  |
| --- | --- |
| $( document ).ready(**function**() {  console.log( "ready!" );  }); | For example: In JQuery We don’t say browser to execute the function till the document gets loaded . Once its gets loaded we ask the browser to execute the callback function.  **function**() {  console.log( "ready!" );  }  **IS A CALLBACK FUNCTION** |

This is called a **CALLBACK PATTERN** of Async java script programming. But we have a problem with callback pattern as well



So the issue here is the callback function in first statement will get executed only when style.css is loaded . Similar in the case of other nested callback, till the function calling nesting reaches the last callback

So Angular come with some other pattern called **PROMISES.**

**“A Promise represent s an operation that hasn’t yet , but expected [to] complete in future ”**

### Log Service

Angular log serviceis used for logging in the application code.

|  |  |
| --- | --- |
| var trainingApp= angular.module("trainingApp",[]);  trainingApp.controller("trainingController", function($log){  $log.info("Code Executed info");  $log.warn("Code Executed with warning");  $log.error("Code Executed with Error");  $log.debug("Code Executed in debug");  $log.log("Log Message");  }); |  |

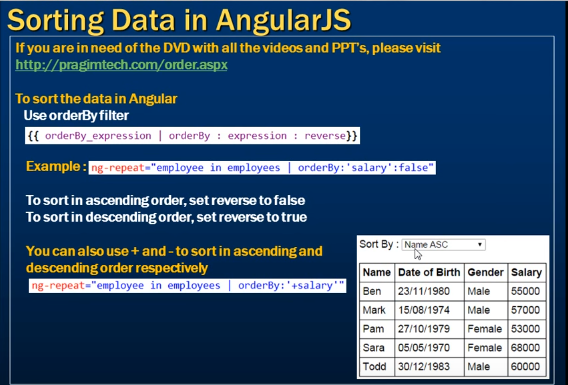
### Formatting using Angular

#### Filter

|  |  |
| --- | --- |
| **FORMATTING IN UPPERCASE**  <p> <input type="text" ng-model="originalTxt"/></p>  <p>{{originalTxt | uppercase}}</p> |  |
| **FORMATTING IN LOWERCASE**  <p> <input type="text" ng-model="originalTxt"/></p>  <p>{{originalTxt | lowercase}}</p> |  |
| **FORMATTING IN CURRENCY**  <p> <input type="text" ng-model="originalTxt"/></p>  <p>{{originalTxt | currency}}</p> |  |
| <p> <input type="text" ng-model="originalTxt"/></p>  <p>{{originalTxt | currency:'Rs.'}}</p> |  |
| **FORMATTING IN CURRENCY[EX. 3 DECIMAL PLACES]**  <p><input type="text" ng-model="originalTxt"/> <span>{{originalTxt | currency:'Rs.':3}}</span> </p> |  |
| **FORMATTING IN NUMBER [EX. 2 DECIMAL PLACES]**  <p><input type="text" ng-model="originalTxt"/> <span>{{originalTxt |number:2}}</span></p> |  |

|  |  |
| --- | --- |
| var employeeMgmtApp = angular.module("employeeMgmtApp",[]);  var employeeArray=[  {firstName :"Sara" , lastName:"Jonshon", salary:5600, dob: new Date("April 12, 1923")},  {firstName :"Mike" , lastName:"Christ", salary:5430, dob: new Date("May 02, 1923")},  {firstName :"John" , lastName:"Hamilton", salary:9600, dob: new Date("July 21, 1923")},  {firstName :"Rebeca" , lastName:"John", salary:8600, dob: new Date("August 24, 1923")},  {firstName :"Dan" , lastName:"Wilson", salary:7500, dob: new Date("June 09, 1923")}  ]  var employeeController = function($scope){  $scope.employee = employeeObj;  $scope.employeeData = employeeArray;  console.log($scope.employee);  $scope.setExperience = function(employee){  employee.exp++;  }  }  employeeMgmtApp.controller("employeeController", employeeController); | |
| <table class="td-table td-table-stripe-column td-table-border-row td-table-border-column" width="100%">  <tbody>  <tr class="td-table-row-first">  <th class="td-table-heading-top">S.no.</th>  <th class="td-table-heading-top">First Name</th>  <th class="td-table-heading-top">Last Name</th>  <th class="td-table-heading-top">Salary(number)</th>  <th class="td-table-heading-top">Salary(Currency)</th>  <th class="td-table-heading-top">DOB</th>  </tr>  <tr ng-repeat="emp in employeeData | **limitTo:2:2**">  <td class="td-copy-sub">{{$index+1}}</td>  <td class="td-copy-sub">{{emp.firstName | uppercase}}</td>  <td class="td-copy-sub">{{emp.lastName |lowercase}}</td>  <td class="td-copy-sub">{{emp.salary | number:2}}</td>  <td class="td-copy-sub">{{emp.salary|currency:'Rs.':1}}</td>  <td class="td-copy-sub">{{emp.dob | date:"dd/MM/yyyy"}}</td>  </tr>  </tbody>  </table> | |
|  | **limitTo:2:2**" 🡪 Means display 2 rows and starting from 2 index(Note : Its zero based index) |

### Sorting



**EXAMPLE**

|  |
| --- |
| **JS**  var employeeMgmtApp = angular.module("employeeMgmtApp",[]);  var employeeArray=[  {firstName :"Sara" , lastName:"Jonshon", salary:5600, dob: new Date("April 12, 1923")},  {firstName :"Mike" , lastName:"Christ", salary:2430, dob: new Date("May 02, 1923")},  {firstName :"John" , lastName:"Hamilton", salary:9600, dob: new Date("July 21, 1923")},  {firstName :"Rebeca" , lastName:"John", salary:10600, dob: new Date("August 24, 1923")},  {firstName :"Dan" , lastName:"Wilson", salary:7500, dob: new Date("June 09, 1923")}  ]  var employeeController = function($scope){  $scope.employeeData = employeeArray;  $scope.sortColumnData ='firstName';  **$scope.reverseSort = false;**  $scope.sortColumn = function(columnToSort){  $scope.sortColumnData = columnToSort;  **$scope.reverseSort = ($scope.sortColumnData == columnToSort) ? !$scope.reverseSort : false;**  }  }  employeeMgmtApp.controller("employeeController", employeeController); |
| **HTML**  <table class="td-table td-table-stripe-column td-table-border-row td-table-border-column" width="100%">  <tbody>  <tr class="td-table-row-first">  <th class="td-table-heading-top">S.no.</th>  <th class="td-table-heading-top" ng-click="sortColumn('firstName')">First Name</th>  <th class="td-table-heading-top" ng-click="sortColumn('lastName')">Last Name</th>  <th class="td-table-heading-top" ng-click="sortColumn('salary')">Salary(number)</th>  <th class="td-table-heading-top" ng-click="sortColumn('salary')">Salary(Currency)</th>  <th class="td-table-heading-top" ng-click="sortColumn('dob')">DOB</th>  </tr>  <tr ng-repeat="emp in employeeData | orderBy:sortColumnData:reverseSort">  <td class="td-copy-sub">{{$index+1}}</td>  <td class="td-copy-sub">{{emp.firstName | uppercase}}</td>  <td class="td-copy-sub">{{emp.lastName |lowercase}}</td>  <td class="td-copy-sub">{{emp.salary | number:2}}</td>  <td class="td-copy-sub">{{emp.salary|currency:'Rs.':1}}</td>  <td class="td-copy-sub">{{emp.dob | date:"dd/MM/yyyy"}}</td>  </tr>  </tbody>  </table> |

#### Searching

|  |
| --- |
| **SORT BY FIRSTNAME**  <div class="td-layout-row">  <div class="td-layout-column td-layout-grid9 td-layout-column-first">  Search: <input type="text" **ng-model="searchTxt.firstName"**/>  <table class="td-table td-table-stripe-column td-table-border-row td-table-border-column" width="100%">  <tbody>  <tr class="td-table-row-first">  <th class="td-table-heading-top">S.no.</th>  <th class="td-table-heading-top">First Name</th>  <th class="td-table-heading-top">Last Name</th>  <th class="td-table-heading-top">Salary(number)</th>  <th class="td-table-heading-top">Salary(Currency)</th>  <th class="td-table-heading-top">DOB</th>  </tr>  <tr ng-repeat="emp in employeeData **| filter:searchTxt"**>  <td class="td-copy-sub">{{$index+1}}</td>  <td class="td-copy-sub">{{emp.firstName | uppercase}}</td>  <td class="td-copy-sub">{{emp.lastName |lowercase}}</td>  <td class="td-copy-sub">{{emp.salary | number:2}}</td>  <td class="td-copy-sub">{{emp.salary|currency:'Rs.':1}}</td>  <td class="td-copy-sub">{{emp.dob | date:"dd/MM/yyyy"}}</td>  </tr>  </tbody>  </table>  </div>  </div> |
|  |

### ng-include

This will include another file in a file.But will work in local host

|  |  |
| --- | --- |
| <div ng-include="'header.html'"></div> | * Enclose the file name in single quotes and then in double quotes. * File(header.html) is included by making a GET request to header.html * It helps to segregate the file in multiple file. |

### Routing

The ngRoute module helps your application to become a Single Page Application.

**What is Routing in AngularJS?**

If you want to navigate to different pages in your application, but you also want the application to be a SPA (Single Page Application), with no page reloading, you can use the ngRoute module.

The ngRoute module routes your application to different pages without reloading the entire application.

**Example:**

|  |  |
| --- | --- |
| **HTML**  <body ng-app="myApp"> <p><a href="#/">Main</a></p> <a href="#red">Red</a> <a href="#green">Green</a> <a href="#blue">Blue</a> <div ng-view></div> | **JS**  var app = angular.module("myApp", ["ngRoute"]); app.config(function($routeProvider) {     $routeProvider     .when("/", {         templateUrl : "main.htm"     })     .when("/red", {         templateUrl : "red.htm"     })     .when("/green", {         templateUrl : "green.htm"     })     .when("/blue", {         templateUrl : "blue.htm"     }); }); |

Navigate to "red.htm", "green.htm", and "blue.htm":

**What do I Need?**

To make your applications ready for routing, you must include the AngularJS Route module:

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.8/angular-route.js"></script>

Then you must add the ngRoute as a dependency in the application module::

var app = angular.module("myApp", ["ngRoute"]);

Now your application has access to the route module, which provides the $routeProvider.

Use the $routeProvider to configure different routes in your application as shown in above example

**Where Does it Go?**

Your application needs a container to put the content provided by the routing. This container is the ng-view directive.

There are three different ways to include the ng-view directive in your application:

1. <div ng-view></div>
2. <ng-view></ng-view>
3. <div class="ng-view"></div>

Applications can only have one ng-view directive, and this will be the placeholder for all views provided by the route.

**$routeProvider 🡺** With the $routeProvider you can define what page to display when a user clicks a link as in above

Define the $routeProvider using the config method of your application. Work registered in the config method will be performed when the application is loading.

**Controllers**

With the $routeProvider you can also define a controller for each "view".

Example: Add controllers:

|  |  |
| --- | --- |
| var app = angular.module("myApp", ["ngRoute"]); app.config(function($routeProvider) {     $routeProvider     .when("/", {         templateUrl : "main.htm"     })     .when("/london", {         templateUrl : "london.htm",         controller : "londonCtrl"     })     .when("/paris", {         templateUrl : "paris.htm",         controller : "parisCtrl"     }); }); | **CONTROLLERS**  app.controller("londonCtrl", function ($scope) {     $scope.msg = "I love London"; }); app.controller("parisCtrl", function ($scope) {     $scope.msg = "I love Paris"; }); |

The "london.htm" and "paris.htm" are normal HTML files, which you can add AngularJS expressions as you would with any other HTML sections of your AngularJS application. The files looks like this:

|  |  |
| --- | --- |
| [london.htm](http://www.w3schools.com/angular/london.htm)  <h1>London</h1> <h3>London is the capital city of England.</h3> <p>It is the most populous city in the United Kingdom, with a metropolitan area of over 13 million inhabitants.</p> <p>{{msg}}</p> | [paris.htm](http://www.w3schools.com/angular/paris.htm)  <h1>Paris</h1> <h3>Paris is the capital city of France.</h3> <p>The Paris area is one of the largest population centers in Europe, with more than 12 million inhabitants.</p> <p>{{msg}}</p> |

**Template**

In the previous examples we have used the templateUrl property in the $routeProvider.when method.

You can also use the template property, which allows you to write HTML directly in the property value, and not refer to a page.

Example:

Write templates:

var app = angular.module("myApp", ["ngRoute"]);  
app.config(function($routeProvider) {  
    $routeProvider  
    .when("/", {  
        template : "<h1>Main</h1><p>Click on the links to change this content</p>"  
    })  
    .when("/banana", {  
        template : "<h1>Banana</h1><p>Bananas contain around 75% water.</p>"  
    })  
    .when("/tomato", {  
        template : "<h1>Tomato</h1><p>Tomatoes contain around 95% water.</p>"  
    });  
});  
**The otherwise method**

In the previous examples we have used the when method of the $routeProvider. You can also use the otherwise method, which is the default route when none of the others get a match.

Example:

If neither the "Banana" nor the "Tomato" link has been clicked, let them know:

var app = angular.module("myApp", ["ngRoute"]);  
app.config(function($routeProvider) {  
   $routeProvider  
    .when("/banana", {  
        template : "<h1>Banana</h1><p>Bananas contain around 75% water.</p>"  
    })  
    .when("/tomato", {  
        template : "<h1>Tomato</h1><p>Tomatoes contain around 95% water.</p>"  
    })  
    .otherwise({  
        template : "<h1>None</h1><p>Nothing has been selected</p>"  
    });  
});

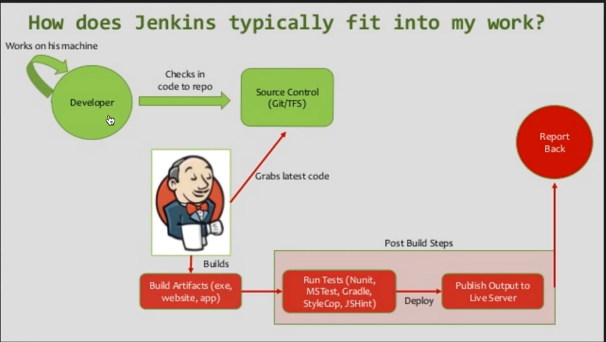
### Project Structure

|  |  |
| --- | --- |
|  | **FILE STRUCTURE** |

|  |
| --- |
| **CREATE A JS FILE TO CREATE ANGULAR MODULE**  function getAngularApp(){  var angularApp;  try {  angularApp = angular.module("employeeAngularApp");  } catch(error){  angularApp = angular.module("employeeAngularApp", []);  }  return angularApp;  } |
| **SERVICE**  var angularApp = getAngularApp();  var getFirstNameFn = function(){  return "Amit";  }  var getLastNameFn = function(){  return "Sinha";  }  angularApp.service("employeeService",function(){  return({  getFirstName: getFirstNameFn, getLastName: getLastNameFn  });  }); |
| **FACTORY**  var angularApp = getAngularApp();  angularApp.factory('employeeFactory',[**'employeeService'**,function(employeeService){  var employeeFactory = {};  console.log(employeeService);  employeeFactory.getFirstName= function(){  return employeeService.getFirstName();  };    employeeFactory.getLastName=function(){  return employeeService.getLastName();  };  return employeeFactory;  }]); |
| **CONTROLLER**  var angularApp = getAngularApp();  angularApp.controller("employeeController",['employeeFactory','$scope',function(employeeFactory,$scope){  console.log(employeeFactory);  $scope.firstName = employeeFactory.getFirstName();  $scope.lastName = employeeFactory.getLastName();  }]); |
| **VIEW[HTML]**  <html ng-app="employeeAngularApp">  <head>  <link rel="stylesheet" href="css/default.css">  <script src="js/angular.js"></script>  <script src="module/applicationModule.js"></script>  <script src="services/employee-service.js"></script>  <script src="factories/employee-factory.js"></script>  <script src="controller/employee-controller.js"></script>  </head>  <body >  <div id="td-wrapper" ng-controller="employeeController">  <div id="td-container">  <div class="td-layout-row" >  <p ng-bind="firstName"></p>  <p ng-bind="lastName"></p>  </div>  </div>  </div>  </body>  </html> |

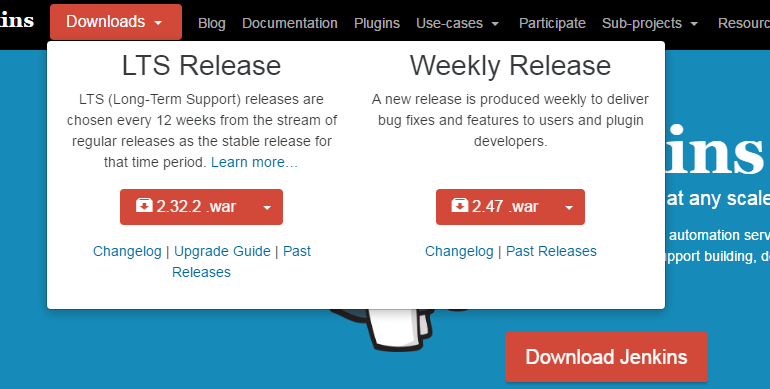
## Jenkins

* It’s a Java Application
* Used for continuous integration and delivery



### Jenkins Setup

Step 1: Download Jenkins war [https://jenkins.io/] – LTS Release



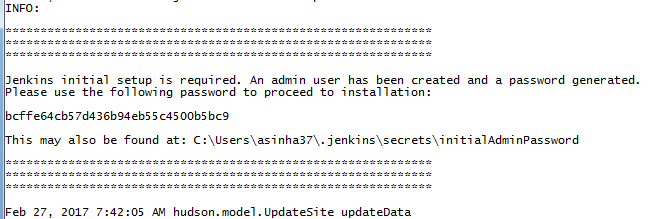
Step 2: Place it in a folder and navigate to the folder using command prompt

Step 3: Unjar the Jenkins war file [java -jar jenkin.war]

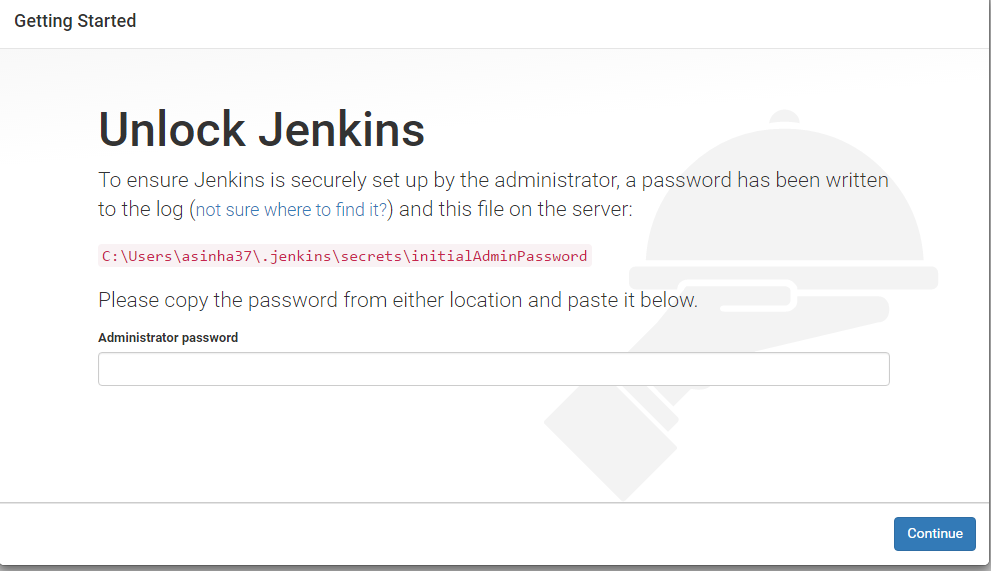
Step 4: After the Jenkins in unjared- The jenkin can be accessed using browser on port -8080 🡪 [http://localhost:8080]

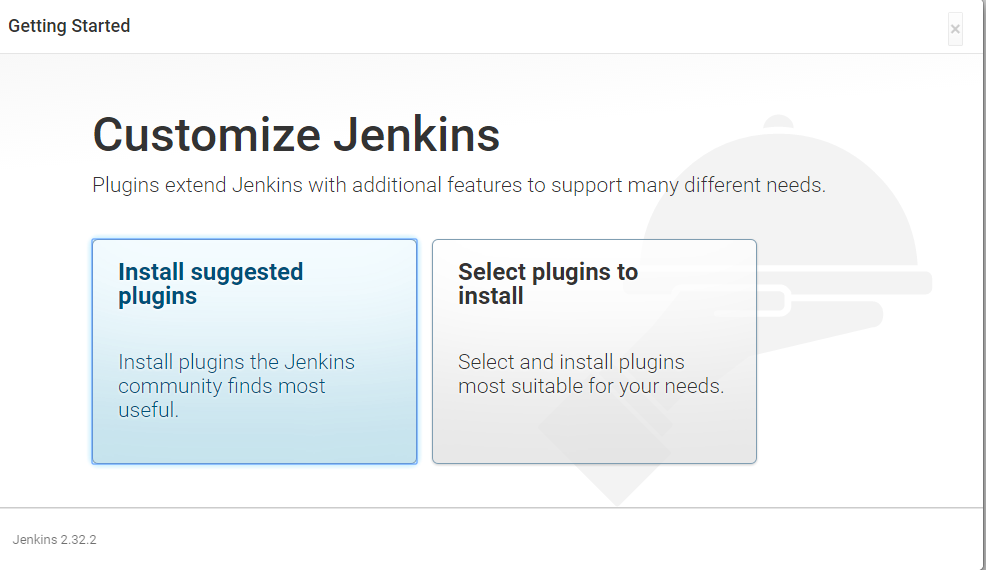
Step 5: When Jenkins is started, it will give a password for admin login in Jenkins as shown below

Note: This password is also saved in a file: [C:\Users\asinha37\.jenkins\secrets\initialAdminPassword]

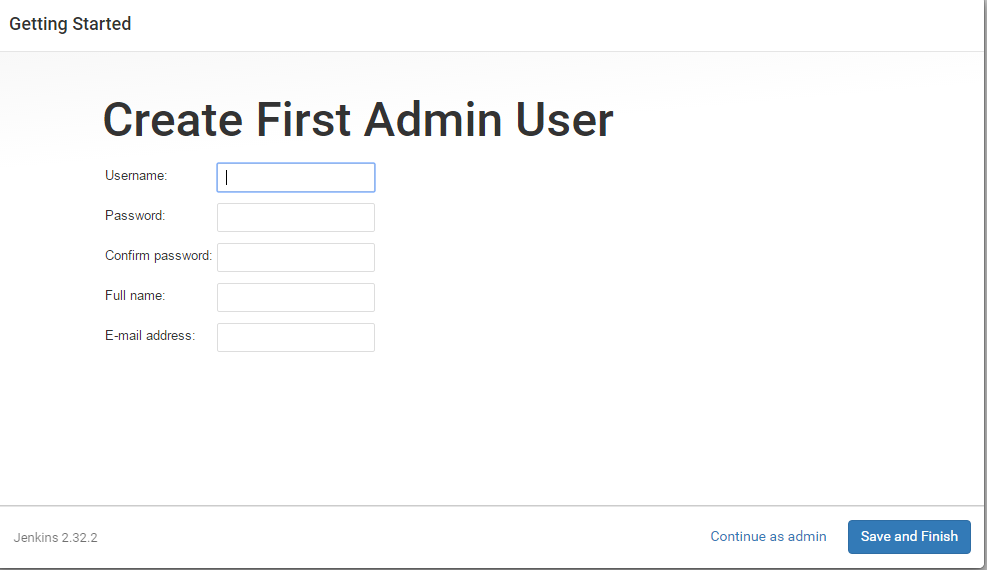


Step 4: After login it will ask to install plugin as shown below[Select an option of suggested plugin]





Continue as admin



#### How to change the home directory of Jenkins

The Jenkins home directory contains

1. All Configurations
2. Logs
3. Job Details
4. Plugins

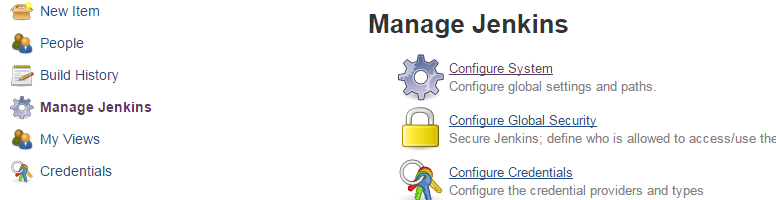
Why we change the home directory of Jenkins?

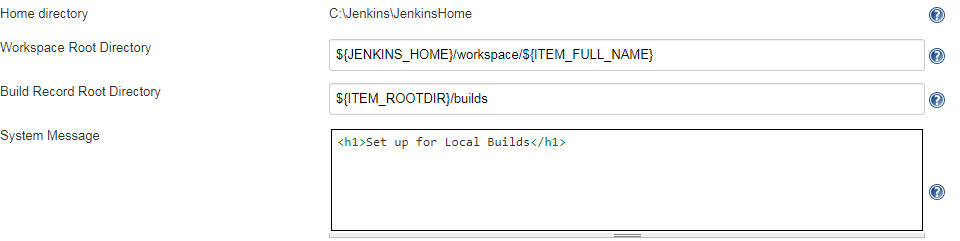
The Jenkins home directory by default gets created in the User Profile folder, so to change the default location of Jenkins to the folder where we have enough disk space.

Step 1: Create a new folder which we want to make a home directory

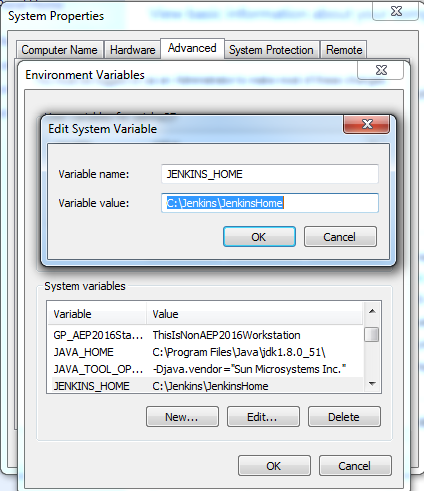
Step 2: Copy the files from the default directory - In user profile (created at the time of installation) to new home directory

Step 3: Go to Manage Jenkin 🡪 Configure System. This will show the default home directory of Jenkins. To change this default home directory follow the below steps

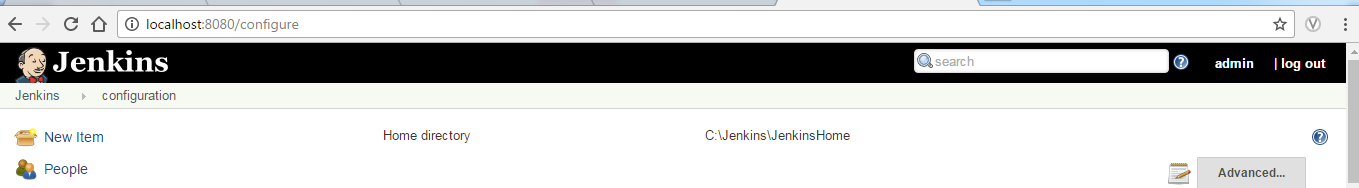




Step 4: Create an Environment variable – JENKINS\_HOME . Restart Jenkins from the command prompt



Home Directory Updated

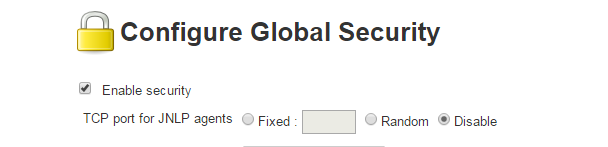


## How to use CLI(Command Line Interface in Jenkins)

Step 1: Go to Manage Jenkins 🡪 Configure Global Security



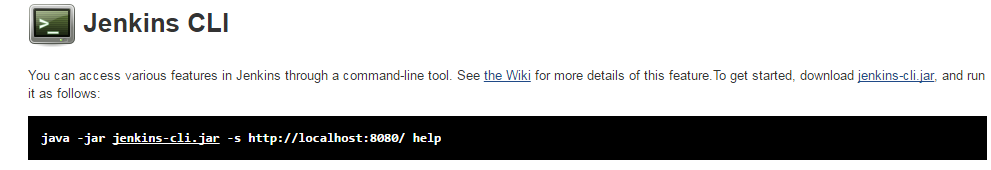
Step 2: Make sure that “Enable Security” checkbox is checked



Step 3: Go to CLI Interface: <http://localhost:8080/cli/>

CLI Wiki: <https://wiki.jenkins-ci.org/display/JENKINS/Jenkins+CLI>

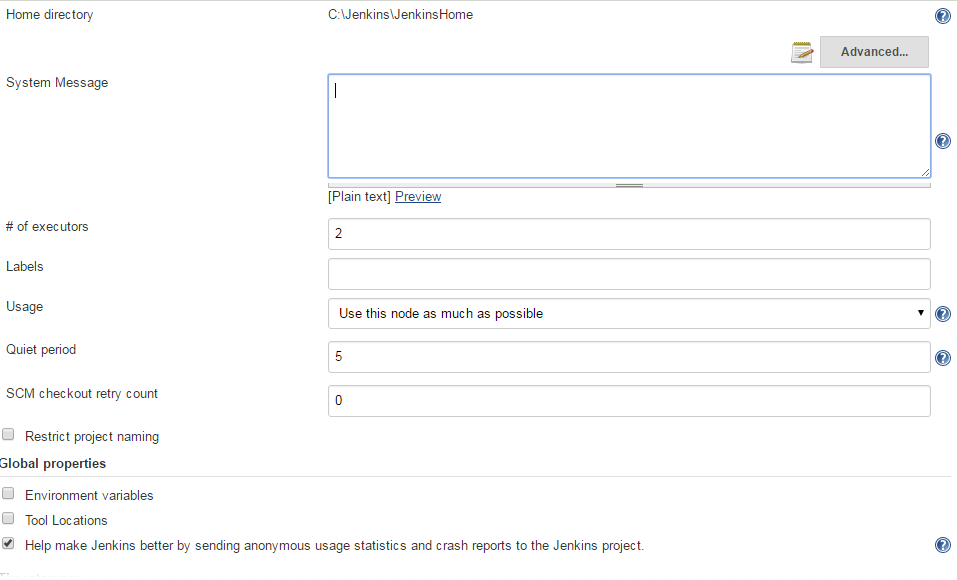
Download the [jenkins-cli.jar](http://localhost:8080/jnlpJars/jenkins-cli.jar) and place it in a desired folder



## Jenkins: Basic Configuration

Step 1: Go to Manage Jenkins🡪Configure System

Step 2: Configuration Details



|  |  |  |
| --- | --- | --- |
| Home directory | Home Directory of Jenkins | |
| System Message (Used as Notification for User) | This message will appear in the home screen of Jenkins | |
|  | | |
| We can also use HTML tags also in System Message  Step 1: Go to Manage Jenkins 🡪 Configure Global Security 🡪 Change Markup formatter from Plaintext to Safe HTML | | |
| Now Use HTML Code in System Message and Save | | |
|  | | |
| # of executors(number of executor) | | This gives the number of JOBS that Jenkins can execute parallelly |

## Jenkins Jobs

|  |  |
| --- | --- |
| 1. Go to jenkins and Click on New Item 2. Enter the name of the JOB and Select the project type as “Free Style Project” 🡪OK 3. Now we can give the details/configuration of the JOB 4. It has 6 tabs in it for specific configurations(described below) |  |

|  |  |
| --- | --- |
| **Tab** | **Description** |
| General |  |
| Source Code Management |  |
| Build Triggers |  |
| Build Environment |  |
| Build |  |
| Post Build Actions |  |

**Chaining Builds**

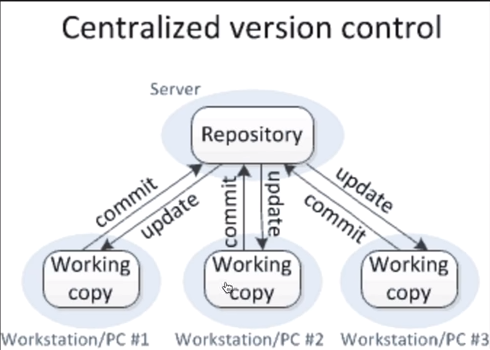
# GIT

It’s a version control system.

Types of VCS

1. Centralized VCS
2. Distributed VCS

Centralized VCS



Distributed VCS (Git)

