## What is HTML?

HTML is a computer language devised to allow website creation.

The definition of HTML is **HyperText Markup Language**.

- **HyperText** is the method by which you move around on the web — by clicking on special text called hyperlinks which bring you to the next page. The fact that it is hyper just means it is not linear — i.e. you can go to any place on the Internet whenever you want by clicking on links — there is no set order to do things in.
- **Markup** is what HTML tags do to the text inside them. They mark it as a certain type of text (italicised text, for example).
- HTML is a **Language**, as it has code-words and syntax like any other language.

## History of HTML

| | |
|---|---|
| 1991 | HTML First published |
| 1995 | HTML 2.0 |
| 1997 | HTML 3.2 |
| 1999 | HTML 4.01 (Now the focus shifted to XHTML and it has stricted standards) |
| 2000 | XHTML 1.0 |
| 2002 - 2009 | XHTML 2.0 (Stricter standards that 1.0, rejected the web pages that did not comply) |
| 2012 | HTML 5 (This much more tolerant, and can handle markup from all the prior versions) |

XHTML stands for EXtensible HyperText Markup Language

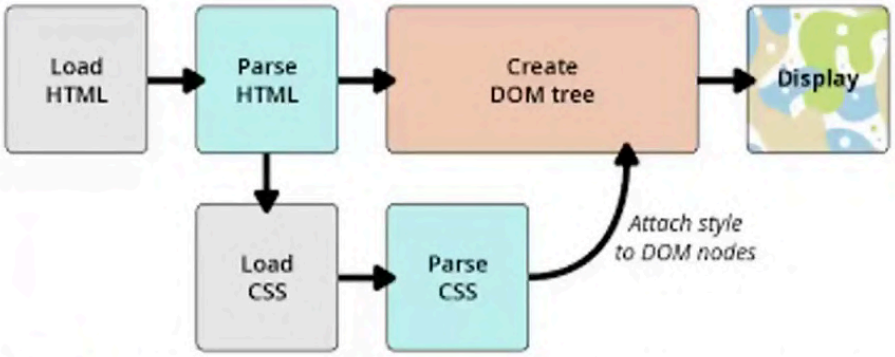
## How HTML work?

- HTML consists of a series of short codes typed into a text-file — these are the tags.
- The text is then saved as a html file, and viewed through a browser,

like Internet Explorer or Netscape Navigator.

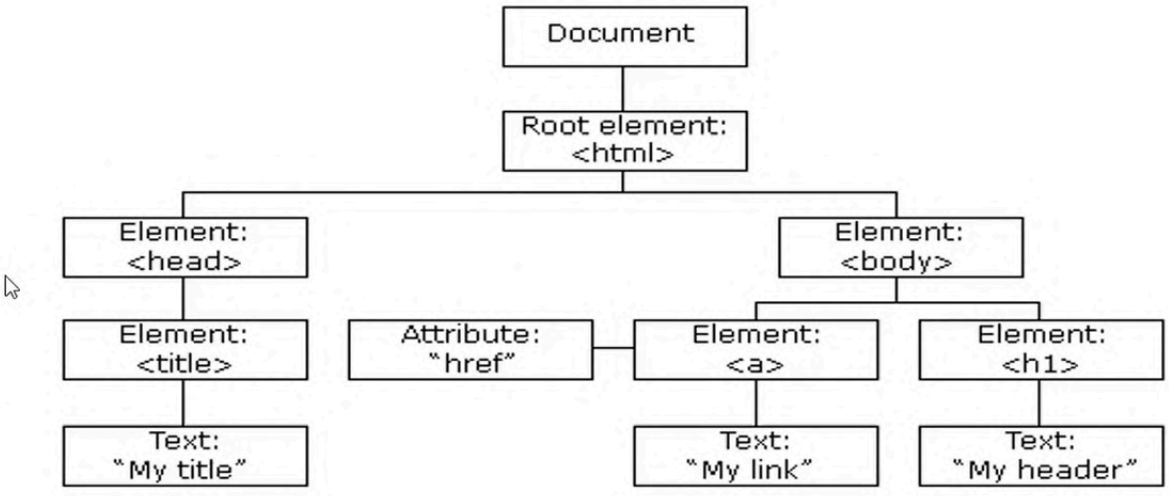
- This browser reads the file and translates the text into a visible form,

hopefully rendering the page as the author had intended.




The **Document Object Model** is a cross-platform and language-independent application programming interface that treats an HTML, XHTML, or XML document as a tree structure wherein each node is an object representing a part of the document.

## HTML DOM Tree



**DOM** stands for **Document Object Model**. It is a programming interface for web documents like HTML, XML, and SVG. The DOM represents the structure of a document as a tree of objects, allowing developers to access, manipulate, and modify the document's elements using scripting languages such as JavaScript.
When a web page is loaded, the browser creates a DOM of the page. This model is constructed as a hierarchical tree of objects, where each node represents an element or attribute in the document. The DOM provides a standard way to interact with these elements, enabling dynamic content updates and interactivity.

## HTML editors

Commonly used HTML editors in the industry are;

- **Coffee Cup HTML editor**
- **Adobe Dreamweaver**
- **Microsoft FrontPage**
- **RapidWeaver**
- **Mozilla KompoZer**

We can achieve the same using **Notepad** or **Notepad++**

## First HTML Page

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
           "http://www.w3.org/TR/html4/strict.dtd">
<html>
    <head>
    ...
    </head>

    <body>
    ...
    </body>
</html>
```

## HTML Elements

An HTML element usually consists of a start tag and end tag, with the content inserted in between:

**<tagname>**Content goes here...**</tagname>**

```
<p>My first paragraph.</p>
```

## HTML Attributes

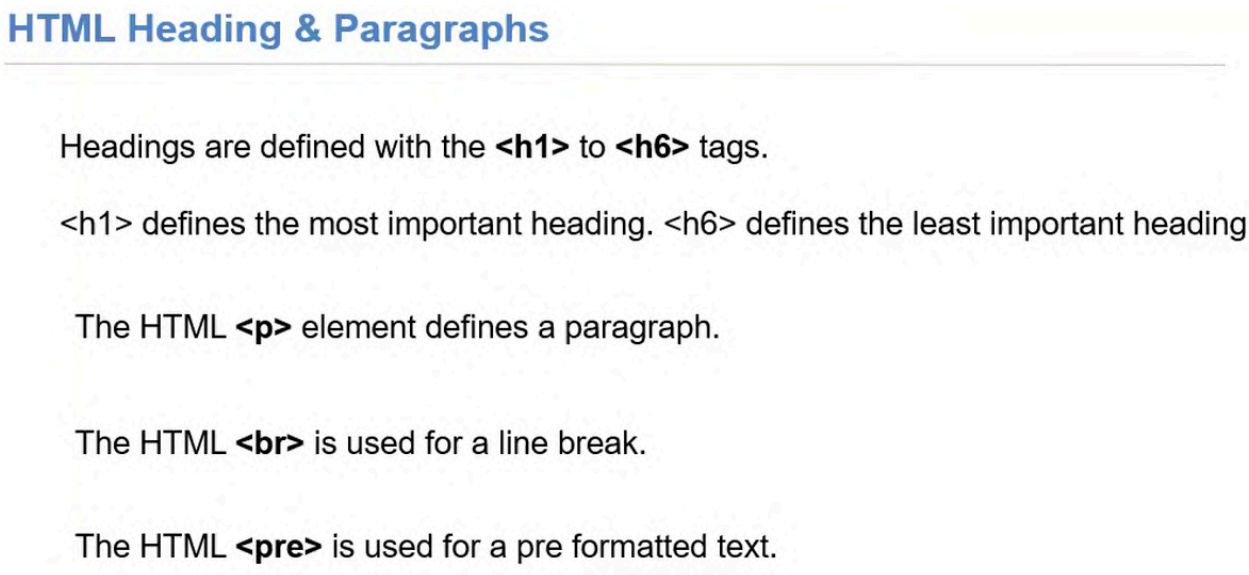
Attributes provide additional information about HTML elements.

- All HTML elements can have attributes
- Attributes provide additional information about an element
- Attributes are always specified in the start tag
- Attributes usually come in name/value pairs like: name="value"

```
<a href="first.html">My First Page</a>
```

**Tag**: The element name in HTML, like `<p>`, `<img>`, or `<div>`. It defines **what** is being created.

**Attribute**: Adds extra **information or behavior** to a tag. It's written inside the opening tag, like `src` in `<img src="image.jpg">`.




### HTML Structure Tags

| Tag | Description | Example |
|---|---|---|
| `<html>` | Root element of an HTML document | `<html> ... </html>` |
| `<head>` | Contains meta info (not visible on page) | `<head> ... </head>` |
| `<title>` | Sets browser tab title | `<title>My Website</title>` |

| | | |
|---|---|---|
| `<body>` | Main content of the page | `<body> ... </body>` |
| `<meta>` | Metadata (like charset, author, etc.) | `<meta charset="UTF-8">` |
| `<link>` | Links external resources (e.g. CSS) | `<link rel="stylesheet" href="style.css">` |
| `<style>` | Embeds internal CSS | `<style>p {color: red;}</style>` |
| `<script>` | Embeds or links JavaScript | `<script src="main.js"></script>` |
| `<base>` | Sets base URL for all relative links | `<base href="https://example.com/">` |

**Text Formatting Tags**

| Tag | Description | Example |
|---|---|---|
| `<h1>` to `<h6>` | Headings from biggest (h1) to smallest (h6) | `<h1>Main Heading</h1>` |
| `<p>` | Paragraph | `<p>This is a paragraph.</p>` |

| | | |
|---|---|---|
| `<br>` | Line break | `Line1<br>Line2` |
| `<hr>` | Horizontal line | `<hr>` |
| `<pre>` | Preformatted text | `<pre> line1\n line2</pre>` |
| `<blockquote>` | Long quote from another source | `<blockquote>Quoted text</blockquote>` |
| `<q>` | Inline short quote | `<q>Quote</q>` |
| `<abbr>` | Abbreviation with tooltip | `<abbr title="World Health Organization">WHO</abbr>` |
| `<cite>` | Citation (e.g., for books, articles) | `<cite>Hamlet</cite>` |
| `<code>` | Displays code | `<code>print("Hello")</code>` |
| `<em>` | Emphasizes text (italic) | `<em>important</em>` |
| `<strong>` | Strong emphasis (bold) | `<strong>very important</strong>` |
| `<mark>` | Highlights text | `<mark>highlighted</mark>` |

| | | |
|---|---|---|
| `<small>` | Smaller text | `<small>fine print</small>` |
| `<del>` | Deleted text (strikethrough) | `<del>old text</del>` |
| `<ins>` | Inserted/added text (underlined) | `<ins>new text</ins>` |
| `<sub>` | Subscript | `H<sub>2</sub>O` |
| `<sup>` | Superscript | `x<sup>2</sup>` |

**List Tags**

| Tag | Description | Example |
|---|---|---|
| `<ul>` | Unordered (bulleted) list | `<ul><li>Item 1</li><li>Item 2</li></ul>` |
| `<ol>` | Ordered (numbered) list | `<ol><li>First</li><li>Second</li></ol>` |
| `<li>` | List item | `<li>Item</li>` |

| | | |
|---|---|---|
| `<dl>` | Description list | `<dl><dt>HTML</dt><dd>Markup Language</dd></dl>` |
| `<dt>` | Term in a description list | `<dt>HTML</dt>` |
| `<dd>` | Description of the term | `<dd>HyperText Markup Language</dd>` |

**Image, Media & Embedded Content Tags**

| Tag | Description | Example |
|---|---|---|
| `<img>` | Displays an image | `<img src="img.jpg" alt="Photo">` |
| `<figure>` | Groups media content with a caption | `<figure><img src="cat.jpg"><figcaption>Cat</figcaption></figure>` |
| `<figcaption>` | Caption for `<figure>` | `<figcaption>Figure 1: Cat</figcaption>` |
| `<audio>` | Embeds sound | `<audio controls><source src="sound.mp3"></audio>` |

| | | |
|---|---|---|
| `<video>` | Embeds video | `<video controls><source src="movie.mp4"></video>` |
| `<source>` | Media source for `<audio>`/`<video>` | `<source src="file.mp4" type="video/mp4">` |
| `<embed>` | Embeds external content (e.g., PDF) | `<embed src="file.pdf" width="600" height="500">` |
| `<object>` | Embeds multimedia/object | `<object data="file.swf" type="application/x-shockwave-flash"></object>` |
| `<iframe>` | Inline frame (embed another webpage) | `<iframe src="https://example.com"></iframe>` |

**Table Tags**

| Tag | Description | Example |
|---|---|---|
| `<table>` | Creates a table | `<table> ... </table>` |

| | | |
|---|---|---|
| `<tr>` | Table row | `<tr> ... </tr>` |
| `<th>` | Table header cell (bold/centered) | `<th>Header</th>` |
| `<td>` | Table data cell | `<td>Data</td>` |
| `<thead>` | Group header rows | `<thead><tr><th>Title</th></tr></thead>` |
| `<tbody>` | Group body rows | `<tbody><tr><td>Info</td></tr></tbody>` |
| `<tfoot>` | Group footer rows | `<tfoot><tr><td>End</td></tr></tfoot>` |
| `<caption>` | Table caption/title | `<caption>Monthly Report</caption>` |
| `<col>` | Sets column properties | `<col span="2" style="background:yellow;">` |
| `<colgroup>` | Groups columns | `<colgroup><col span="2"></colgroup>` |

**Form and Input Tags**

| Tag | Description | Example |
| --- | --- | --- |
| `<form>` | Defines a form | `<form action="/submit" method="post">...</form>` |
| `<input>` | Input field | `<input type="text" name="username">` |
| `<label>` | Label for an input | `<label for="user">Username</label>` |
| `<textarea>` | Multi-line text input | `<textarea rows="4" cols="50"></textarea>` |
| `<button>` | Clickable button | `<button type="submit">Submit</button>` |
| `<select>` | Drop-down menu | `<select><option>One</option></select>` |
| `<option>` | Option in drop-down | `<option value="1">One</option>` |
| `<optgroup>` | Group of options | `<optgroup label="Group"><option>Item</option></optgroup>` |

| | | |
|---|---|---|
| `<fieldset>` | Groups related fields | `<fieldset><legend>Info</legend>... </fieldset>` |
| `<legend>` | Title for `<fieldset>` | `<legend>Personal Info</legend>` |
| `<datalist>` | Auto-suggestions | `<datalist id="browsers"><option value="Chrome"></datalist>` |
| `<output>` | Output from calculation | `<output id="result">5</output>` |
| `<input type="submit">` | Submit form | `<input type="submit" value="Send">` |

**Semantic Tags (Meaningful Structure)**

| Tag | Description | Example |
|---|---|---|
| `<header>` | Intro/heading section of page/section | `<header><h1>Welcome</h1></header>` |
| `<footer>` | Footer section of page | `<footer>© 2025</footer>` |

| | | |
|---|---|---|
| `<nav>` | Contains navigation links | `<nav><a href="/">Home</a></nav>` |
| `<main>` | Main content (only once per page) | `<main><h2>Article</h2></main>` |
| `<section>` | Logical section/block of content | `<section><h2>About</h2></section>` |
| `<article>` | Independent piece of content | `<article><h2>News</h2></article>` |
| `<aside>` | Side content like ads, notes | `<aside>Sponsored</aside>` |
| `<figure>` | Groups media and caption | `<figure><img src=""><figcaption>Cat</figcaption></figure>` |
| `<figcaption>` | Caption for `<figure>` | `<figcaption>Figure 1: Cat</figcaption>` |
| `<mark>` | Highlighted text | `<p>This is <mark>important</mark>.</p>` |

| | | |
|---|---|---|
| `<time>` | Time/date | `<time datetime="2025-07-25">Today</time>` |
| `<summary>` | Visible heading for `<details>` | `<summary>Click to expand</summary>` |
| `<details>` | Expandable details box | `<details><summary>More</summary><p>Hidden text</p></details>` |

**Interactive and Script Tags**

| Tag | Description | Example |
|---|---|---|
| `<script>` | Adds JavaScript | `<script>console.log('Hello')</script>` |
| `<noscript>` | Content shown if JavaScript disabled | `<noscript>Please enable JS</noscript>` |
| `<dialog>` | Creates popup/modal dialog | `<dialog open>Hi!</dialog>` |
| `<canvas>` | For drawing via JavaScript | `<canvas width="200" height="100"></canvas>` |
| `<slot>` | Web Components: Slot placeholder | `<slot name="title"></slot>` |

| | | |
|---|---|---|
| `<template>` | Client-side template (not rendered until used by JS) | `<template id="myTemp">Hello</template>` |

### Layout & Container Tags

| Tag | Description | Example |
|---|---|---|
| `<div>` | Generic container (block) | `<div class="box"></div>` |
| `<span>` | Generic container (inline) | `<span class="label">Info</span>` |
| `<br>` | Line break | `Line 1<br>Line 2` |
| `<hr>` | Thematic break (horizontal line) | `<hr>` |
| `<wbr>` | Optional line break point | `long<wbr>word` |

### Global HTML Attributes (used with *any* tag)

| Attribute | Description | Example |
|---|---|---|

| | | |
|---|---|---|
| `id` | Unique identifier | `<div id="header">` |
| `class` | CSS class name(s) | `<p class="text-bold">` |
| `style` | Inline CSS styling | `<h1 style="color:red">` |
| `title` | Tooltip text | `<abbr title="HyperText Markup Language">HTML</abbr>` |
| `hidden` | Hides element | `<div hidden>This won't show</div>` |
| `lang` | Language of element | `<html lang="en">` |
| `dir` | Text direction (`ltr` or `rtl`) | `<p dir="rtl">مرحبا</p>` |
| `tabindex` | Tab key navigation order | `<input tabindex="1">` |
| `contenteditable` | Makes element editable | `<div contenteditable="true">Edit me</div>` |
| `draggable` | Makes element draggable | `<img draggable="true">` |

| | | |
|---|---|---|
| `accesskey` | Keyboard shortcut | `<button accesskey="s">Save</button>` |
| `data-*` | Custom data attributes | `<div data-user-id="123">` |

## Styling HTML with CSS

CSS stands for Cascading Style Sheets.

CSS describes how HTML elements are to be displayed on screen, paper, or in other media.

CSS saves a lot of work. It can control the layout of multiple web pages all at once

CSS can be added to HTML elements in 3 ways:

- **Inline** - by using the style attribute in HTML elements
- **Internal** - by using a <style> element in the <head> section
- **External** - by using an external CSS file

The most common way to add CSS, is to keep the styles in separate CSS files. However,
here we will use inline and internal styling, because this is easier to demonstrate, and easier for you to try it yourself.

## What is HTML5

**HTML5** is the latest version of Hypertext Markup Language, the code that describes web pages. It's actually three kinds of code:

- HTML, which provides the structure;
- Cascading Style Sheets (CSS), which take care of presentation
- JavaScript, which makes things happen.


HTML5 has been designed to deliver almost everything you'd want to do online without requiring additional software such as browser plugins.


## HTML & HTML5

| HTML | HTML5 |
|---|---|
| It didn't support audio and video without the use of flash player support. | It supports audio and video controls with the use of <audio> and <video> tags. |
| It uses cookies to store temporary data. | It uses SQL databases and application cache to store offline data. |
| Does not allow JavaScript to run in browser. | Allows JavaScript to run in background. This is possible due to JS Web worker API in HTML5. |
| Vector graphics is possible in HTML with the help of various technologies such as VML, Silver-light, Flash, etc. | Vector graphics is additionally an integral a part of HTML5 like SVG and canvas. |
| It does not allow drag and drop effects. | It allows drag and drop effects. |
| Not possible to draw shapes like circle, rectangle, triangle etc. | HTML5 allows to draw shapes like circle, rectangle, triangle etc. |
| It works with all old browsers. | It supported by all new browser like Firefox, Mozilla, Chrome, Safari, etc. |

## HTML5 Elements

New Elements introduced in HTML5 are:

1) **Semantic elements** : A semantic element clearly describes its meaning to both the browser and the developer.
<header>, <footer>, <article>, and <section>.

2) **Form elements**
number, date, time, calendar, and range.

3) **Graphic elements**
<svg> and <canvas>.

4) **Multimedia elements:**
<audio> and <video>.

### 1. Color & Background Properties

| Property | Description | Example |
|---|---|---|
| `color` | Text color | `color: red;` |
| `background-color` | Background color | `background-color: lightblue;` |
| `background-image` | Background image | `background-image: url("bg.jpg");` |
| `background-size` | Size of background image | `background-size: cover;` |

| | | |
|---|---|---|
| `background-repeat` | Repeat background image | `background-repeat: no-repeat;` |
| `background-position` | Position of background image | `background-position: center top;` |
| `background` | Shorthand for background properties | `background: #fff url('img.jpg') no-repeat right top;` |
| `opacity` | Transparency level (0 to 1) | `opacity: 0.5;` |

**2. Box Model (Margin, Border, Padding, Width, Height)**

| Property | Description | Example |
|---|---|---|
| `margin` | Outer space of element | `margin: 10px;` |
| `margin-top/bottom/left/right` | Margin on specific side | `margin-left: 20px;` |
| `padding` | Inner space between content and border | `padding: 15px;` |

| | | |
|---|---|---|
| `padding-top/bottom/left/right` | Padding on specific side | `padding-top: 10px;` |
| `border` | Border around element | `border: 1px solid black;` |
| `border-radius` | Rounded corners | `border-radius: 10px;` |
| `width` | Element width | `width: 300px;` |
| `height` | Element height | `height: 100px;` |
| `box-sizing` | Includes padding & border in width | `box-sizing: border-box;` |

### 3. Positioning & Layout

| Property | Description | Example |
|---|---|---|
| `position` | Positioning method (`static`, `relative`, `absolute`, `fixed`, `sticky`) | `position: absolute;` |
| `top`, `right`, `bottom`, `left` | Offset for positioned element | `top: 20px;` |

| | | |
|---|---|---|
| `z-index` | Stack order (higher = on top) | `z-index: 10;` |
| `display` | Layout type (`block`, `inline`, `flex`, `grid`, `none`) | `display: flex;` |
| `float` | Float element left or right | `float: left;` |
| `clear` | Prevent float overlap | `clear: both;` |
| `overflow` | Scroll or hide overflow | `overflow: auto;` |
| `visibility` | Show/hide element but keep space | `visibility: hidden;` |

### 4. Text & Font Properties

| Property | Description | Example |
|---|---|---|
| `font-family` | Sets the font type | `font-family: Arial, sans-serif;` |
| `font-size` | Sets the text size | `font-size: 16px;` |

| | | |
|---|---|---|
| `font-style` | Sets style (`normal`, `italic`, `oblique`) | `font-style: italic;` |
| `font-weight` | Thickness of font (`normal`, `bold`, `400`, `700`) | `font-weight: bold;` |
| `line-height` | Space between lines | `line-height: 1.5;` |
| `letter-spacing` | Space between letters | `letter-spacing: 2px;` |
| `word-spacing` | Space between words | `word-spacing: 5px;` |
| `text-align` | Horizontal text alignment | `text-align: center;` |
| `text-decoration` | Underline, overline, etc. | `text-decoration: underline;` |
| `text-transform` | Capitalization (`uppercase`, `lowercase`, `capitalize`) | `text-transform: uppercase;` |
| `white-space` | How whitespace is handled | `white-space: nowrap;` |

| | | |
|---|---|---|
| `direction` | Text direction (LTR or RTL) | `direction: rtl;` |

### 5. Flexbox Properties (for layout)

| Property | Description | Example |
|---|---|---|
| `display: flex` | Enables flex container | `display: flex;` |
| `flex-direction` | Row or column layout | `flex-direction: column;` |
| `justify-content` | Main-axis alignment | `justify-content: space-between;` |
| `align-items` | Cross-axis alignment | `align-items: center;` |
| `align-self` | Align single item in container | `align-self: flex-end;` |
| `flex-wrap` | Wrap items on new line | `flex-wrap: wrap;` |
| `flex-grow` | Allow item to grow | `flex-grow: 1;` |
| `flex-shrink` | Allow item to shrink | `flex-shrink: 0;` |

| | | |
|---|---|---|
| `flex-basis` | Initial size of flex item | `flex-basis: 200px;` |
| `gap` | Space between flex items | `gap: 10px;` |

### 6. Grid Layout Properties

| Property | Description | Example |
|---|---|---|
| `display: grid` | Enables grid container | `display: grid;` |
| `grid-template-columns` | Define column sizes | `grid-template-columns: 1fr 2fr;` |
| `grid-template-rows` | Define row sizes | `grid-template-rows: 100px auto;` |
| `grid-gap` or `gap` | Space between grid items | `gap: 20px;` |
| `grid-column` | Positioning grid items in columns | `grid-column: 1 / 3;` |
| `grid-row` | Positioning grid items in rows | `grid-row: 2 / 4;` |
| `place-items` | Align items both ways | `place-items: center;` |

### 7. Animation & Transition Properties

| Property | Description | Example |
|---|---|---|
| `transition` | Applies transition effect | `transition: all 0.3s ease;` |
| `transition-property` | Specifies what to animate | `transition-property: background-color;` |
| `transition-duration` | How long the transition takes | `transition-duration: 0.5s;` |
| `transition-timing-function` | Speed curve | `transition-timing-function: ease-in-out;` |
| `transition-delay` | Delay before start | `transition-delay: 0.2s;` |
| `animation` | Shorthand for animation | `animation: slideIn 1s ease-in;` |
| `animation-name` | Name of keyframes | `animation-name: bounce;` |
| `animation-duration` | Total duration | `animation-duration: 2s;` |

| | | |
|---|---|---|
| `animation-timing-function` | Speed curve | `animation-timing-function: ease-in;` |
| `animation-delay` | Delay before animation starts | `animation-delay: 1s;` |
| `animation-iteration-count` | Number of repeats | `animation-iteration-count: infinite;` |
| `animation-direction` | Forward, reverse, or alternate | `animation-direction: alternate;` |
| `animation-fill-mode` | How styles apply before/after | `animation-fill-mode: forwards;` |
| `animation-play-state` | Running or paused | `animation-play-state: paused;` |

### 8. Transform & Filter Properties

| Property | Description | Example |
|---|---|---|
| `transform` | Applies 2D/3D transformations | `transform: rotate(45deg);` |

| | | |
|---|---|---|
| `transform-origin` | Sets the origin point for transform | `transform-origin: center;` |
| `perspective` | Sets 3D perspective | `perspective: 1000px;` |
| `perspective-origin` | Origin of 3D perspective | `perspective-origin: top left;` |
| `backface-visibility` | Hides back of element in 3D | `backface-visibility: hidden;` |

**Filter Properties**

| Property | Description | Example |
|---|---|---|
| `filter` | Applies graphical effects like blur, brightness | `filter: blur(5px);` |
| `backdrop-filter` | Applies effects to background behind element | `backdrop-filter: brightness(0.8);` |

## 9. Lists, Tables & Form Properties

**List Properties**

| Property | Description | Example |
|---|---|---|
| `list-style` | Shorthand for list styles | `list-style: square inside;` |
| `list-style-type` | Type of bullet or numbering | `list-style-type: circle;` |
| `list-style-position` | Position of bullet/number | `list-style-position: inside;` |
| `list-style-image` | Image as bullet | `list-style-image: url('dot.png');` |

**Table Properties**

| Property | Description | Example |
|---|---|---|
| `border-collapse` | Collapse table borders | `border-collapse: collapse;` |
| `border-spacing` | Space between borders | `border-spacing: 10px;` |

| | | |
|---|---|---|
| `table-layout` | Layout algorithm used for table | `table-layout: fixed;` |
| `caption-side` | Position of caption | `caption-side: bottom;` |
| `empty-cells` | Show/hide borders on empty cells | `empty-cells: hide;` |

**Form Control Properties**

| Property | Description | Example |
|---|---|---|
| `appearance` | Controls the default style of form elements | `appearance: none;` |
| `resize` | Allows or disables textarea resizing | `resize: none;` |
| `outline` | Border outside of element (esp. focus) | `outline: 2px solid blue;` |
| `caret-color` | Color of text input cursor | `caret-color: red;` |
| `accent-color` | Color for checkboxes/radios/sliders | `accent-color: green;` |

| | | |
|---|---|---|
| `::placeholder` | Styling placeholder text | `::placeholder { color: gray; }` |

## What is JavaScript?

**Definition:**

- An object-oriented computer programming language commonly used to create interactive effects within web browsers
- Introduced by Netscape named as ECMAScript later termed as JavaScript. JavaScript has remained the best-known implementation of ECMAScript since the standard was first published
- Other well-known implementations including JScript and ActionScript.
- ECMAScript is commonly used for client-side scripting on the World Wide Web
- It is increasingly being used for writing server applications and services using Node.js
- It is a dynamic computer programming language
- It is a lightweight, interpreted programming language
- Help to create dynamic webpages

## What is JavaScript?

### Versions

~~1.2 4th Edition (abandoned)~~
1.3 5th Edition
1.4 6th Edition - ECMAScript 2015
1.5 7th Edition - ECMAScript 2016
1.6 8th Edition - ECMAScript 2017
1.7 9th Edition - ECMAScript 2018

### Browser Support

Internet Explorer, Mozilla Firefox, Chrome, Opera & Safari

### Prerequisites

Need basic understanding of HTML & CSS.
Also, some prior exposure to object-oriented programming

## Advantages

- **Less server interaction**
  You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.

- **Immediate feedback to the visitors**
  They don't have to wait for a page reload to see if they have forgotten to enter something

- **Increased interactivity**
  You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard

- **Richer interfaces**
  You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

## Limitations

- Client-side JavaScript does not allow the reading or writing of files
- JavaScript doesn't have any multithreading or multiprocessor capabilities.
- Security Issues
- Javascript rendering varies

## JavaScript Development Tools

1. **Visual Studio Code**
2. **Sublime Text Editor**
3. **Microsoft FrontPage**
4. **Macromedia Dreamweaver MX**

# JavaScript - Syntax

```
<html>
<head>
<script  src="../js/Test.js">
</script>
</head>
Code will be written in JS file
<body>
<script type="text/javascript">
  //JS code
</script>
</body>
</html>
```

JSCode can also be written inside the body of script tag.

Or

Can execute using Node

# JavaScript - Syntax

**Case Sensitivity**

Java script is a case sensitive language.

**Semicolon**

var x = 10
var y = 10

or

var x = 10; var y = 10

**Comments**

//
/* */

## JavaScript Datatypes

**Numbers,** eg. 123, 120.50 etc.

**Strings** of text e.g. "This text string" or 'Between single Quotes' etc.

**Boolean** e.g. true or false.

**Custom Objects :** Will encapsulate methods and variables

**Predefined Objects** : Regex, Date, DOM, Math, JSON & Array

## JavaScript Variables

**Variable Names**

var x;

var y = 100;

var _alpha="Test";

var $ = true

Should start with alphabhets, $ or_ . Shouldn't start with numbers and other characters. Don't use reserved key words.

Don't use reserved keywords for variable names like break, Boolean etc.

Variable declaration can be done through keywords **var, let & const**.

Java is type independent

Var can be declared again but let can’t be but you can reassign
And with const we can neither re declare nor reassign

```
var x = 100 ;

var x = “hello” ;   // works

let y = 100 ;

let y = “Hello”;  // error

y  = 100 ;

const z = 100 ;

const z = 200 ; // error

z  = “hello” ;   // error
```

For printing we use

## console .log()


Instance variable : defining variable inside a class

## JavaScript - Operators

Arithmetic Operators (+, -, *, % )

Comparison Operators (==, !=, >=,<=,<,>)

Logical (or Relational) Operators (&&, ||)

Assignment Operators( = , +=, -=)

Conditional (or ternary) Operators (? :)

If Condition is true? Then value X : Otherwise value Y

=== iska mtlb both value and the type should match

## JavaScript - if...else Statement

**if statement**

```
if(condition){
}
```

**if...else statement**

```
if(condition) {
}else{

}
```

**if...else if... statement.**

```
If(condition) {
} else if(condition)
else{
}
```

## Switch Statement

```
switch (expression)
{
  case condition 1: statement(s)
  break;

  case condition 2: statement(s)
  break;
  ...

  case condition n: statement(s)
  break;

  default: statement(s)
}
```

## JavaScript – Loops and Iterations

```
while (expression){
  Statement(s) to be executed if expression is true
}

do{
  Statement(s) to be executed;
} while (expression);


for (initialization; test condition; iteration statement){
  Statement(s) to be executed if test condition is true
}
```

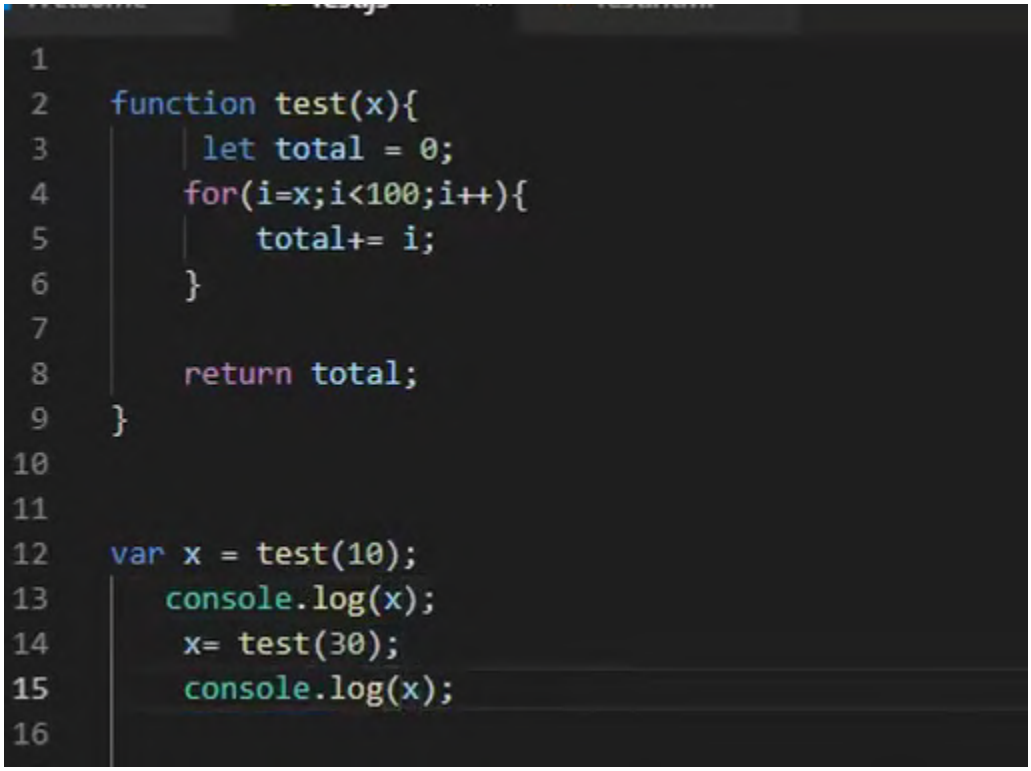
Break and Continue statement  can be used for loop control

## JavaScript - Functions

**Function Definition**

The most common way to define a function in JavaScript is by using the **function** keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

```
<script type="text/javascript">
  <!--
    function <functionname>(parameter-list)
    {
      statements
    }
  //-->
</script>
```



```
1
2   function test(x){
3       let total = 0;
4       for(i=x;i<100;i++){
5           total+= i;
6       }
7
8       return total;
9   }
10
11
12  var x = test(10);
13     console.log(x);
14      x= test(30);
15      console.log(x);
16
```

documents.write (yaha jo bhi likhege vo direct html page p aayega but not in console )

Procedural and structural  language

No need to define the type of the function as javascript is type independent

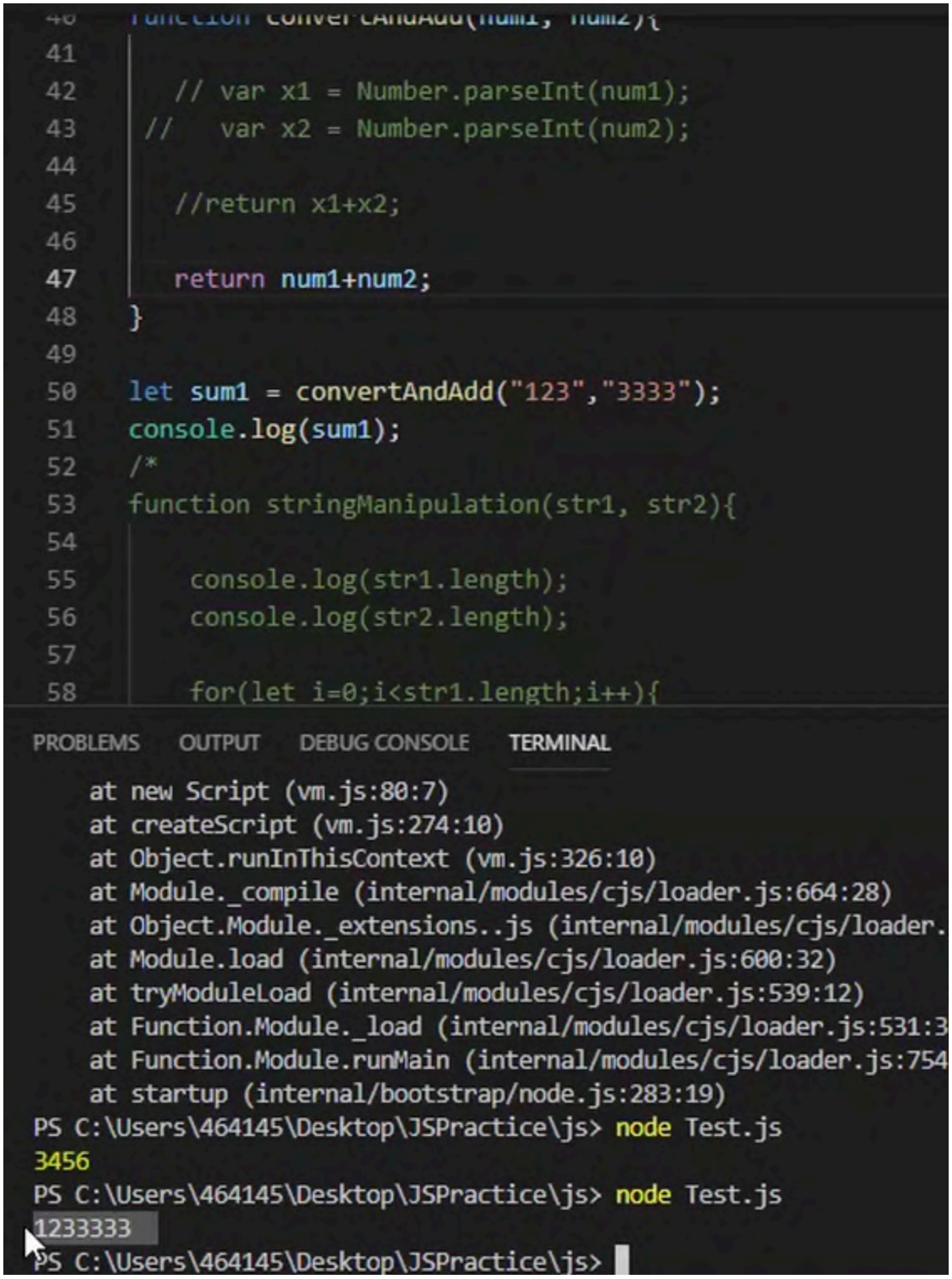
## Java Script- Native objects

The **Number** object represents numerical date, either integers or floating-point numbers

The **Boolean** object represents two values, either "true" or "false"

The **String** object lets you work with a series of characters; it wraps JavaScript's string primitive data type with a number of helper methods.

The **Math** object provides you properties and methods for mathematical constants and functions.




string to integer , use parseInt

# Java Script- String object

**charAt()**

Returns the character at the specified index.

**concat()**

Combines the text of two strings and returns a new string.

**trim()**

Removes the whitespaces from the front and back of the string

**substring()**

Returns the characters in a string between two indexes into the string.

**toLowerCase()**

Returns the calling string value converted to lower case.

**toUpperCase()**

Returns the calling string value converted to uppercase.

```
function stringManipulation(str1, str2){

    console.log(str1.length);
    console.log(str2.length);

    for(let i=0;i<str1.length;i++){
        console.log(str1.charAt(i));
    }

    console.log(str1.concat("    ").concat(str2));

    let newstring = `My First "" '' \ \\\'"""'Name is ${str1} and last name is ${str2}`;



    let sub = str1.substring(0,2);


   // console.log(str1.match());

    console.log(sub);
}
```

## Java Script- Regular Expression

**A regular expression is an object that describes a pattern of characters.**

var pattern = new RegExp(pattern);

or simply

var pattern = /pattern/;

pattern − A string that specifies the pattern of the regular expression or another regular expression.

Eg

var pattern = /^[A-Za-z]+$/;

pattern.test("Ravi");-true

# Java Script- Arrays

**concat()**

Returns a new array comprised of this array joined with other array(s) and/or value(s).

**forEach()**

Calls a function for each element in the array.

**push()**

Adds one or more elements to the end of an array and returns the new length of the array.

**pop()**

Removes the last element from an array and returns that element.

**reverse()**

Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.

**sort()**

Sorts the elements of an array

```
arr.push("Test");
arr.push("Test1");
arr.push("Green");

 //arr.pop();

for(let i=0;i<arr.length;i++){

   console.log(arr[i]);
}

arr.forEach(function(data){
  console.log(data);
});
```

```
arr.push("Test");
arr.push("Test1");
arr.push("Green");

 //arr.pop();

for(let i=0;i<arr.length;i++){

   console.log(arr[i]);
}

arr.forEach(function(data, index){
  console.log(data+" "+index);
});
```

```
113
114         arr.sort(function(x,y){
115             return x>y;
116         })
117
118
119         console.log(arr);
120
121
122     }
123
124     arrayDesc();
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Test1
Green
Test 0
Test1 1
Green 2
PS C:\Users\464145\Desktop\JSPractice\js> node Test.js
Hello
Abc
Green
Hello 0
Abc 1
Green 2
[ 'Abc', 'Green', 'Hello' ]
```


109 if(arr[i]=="Hello"){
110 arr.splice(i,1);
111 break;
112 }
113 }
114
115 /*arr.forEach(function(data, index){
116 console.log(data+" "+index);
117 });
118
119 arr.sort(function(x,y){
120 return x>y;
121 })*/
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Test1 1
Green 2
PS C:\Users\464145\Desktop\JSPractice\js> node Test.js
Hello
Abc
Green
Hello 0
Abc 1
Green 2
[ 'Abc', 'Green', 'Hello' ]
PS C:\Users\464145\Desktop\JSPractice\js> node Test.js
Hello
[ 'Abc', 'Green' ]
PS C:\Users\464145\Desktop\JSPractice\js>

# Java Script- Custom Objects

**Object Creation and Initialization**

```
var fruit = {
    name : "apple",
    shape : "round",
    weightingms : 5
}
console.log(fruit.name);
console.log(fruit.shape);
```

**or**

```
var fruit = new Object();
fruit.name = "Apple";
fruit.shape = "round";
fruit.weightingms = 5;
console.log(fruit.name)
```

```
function customObject(){

    let custom = {

            customerName : "Abc",

            custAddress : "Tvm",

            custAge : 45
    }


    console.log(custom.customerName);
    console.log(custom.custAddress);
    console.log(custom.custAge)
```

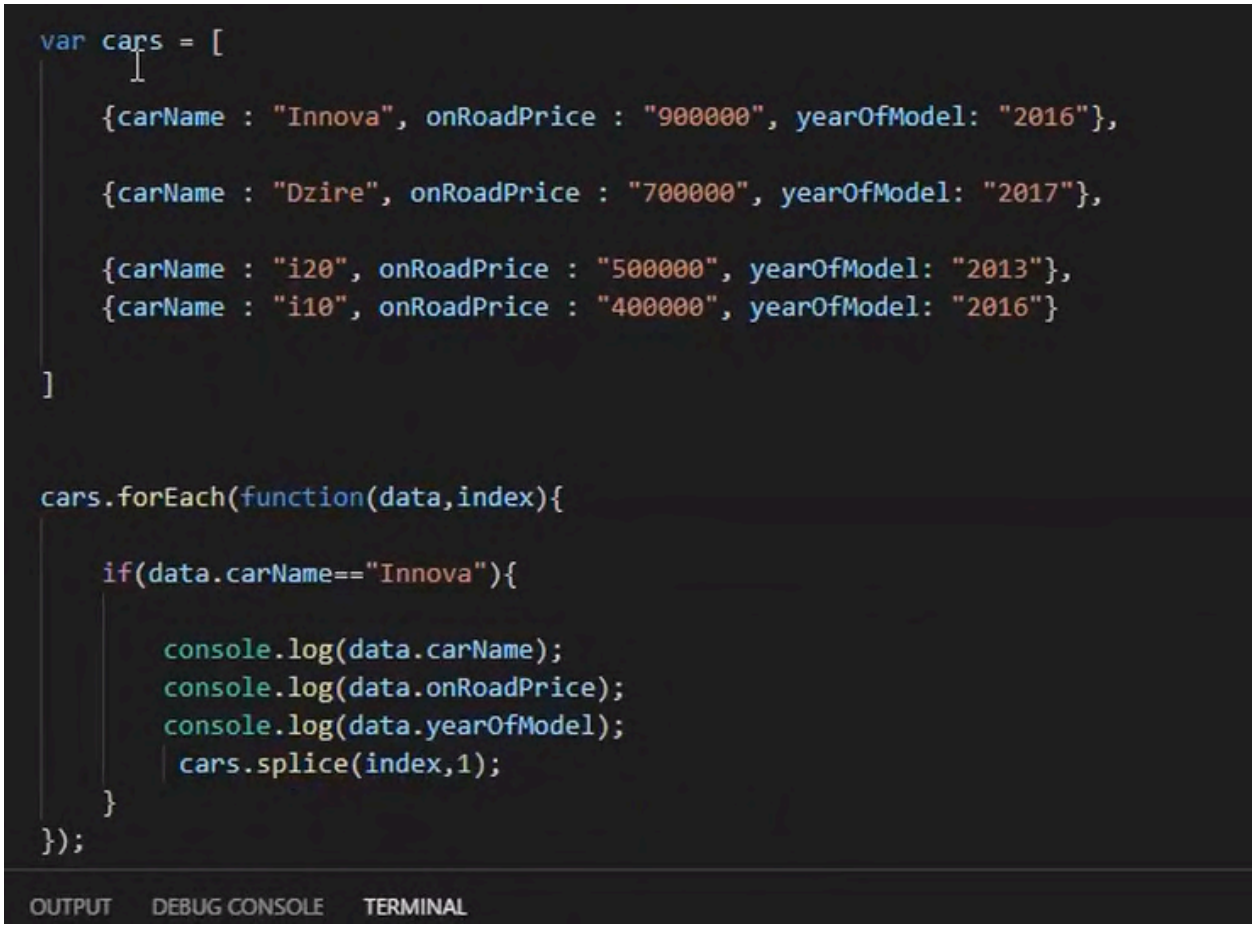
```
    console.log(custom.custAge);

    custom.customerName= "Abc Travels";

    console.log(custom.customerName);
    console.log(custom.custAddress);
    console.log(custom.custAge);


}

customObject();
```

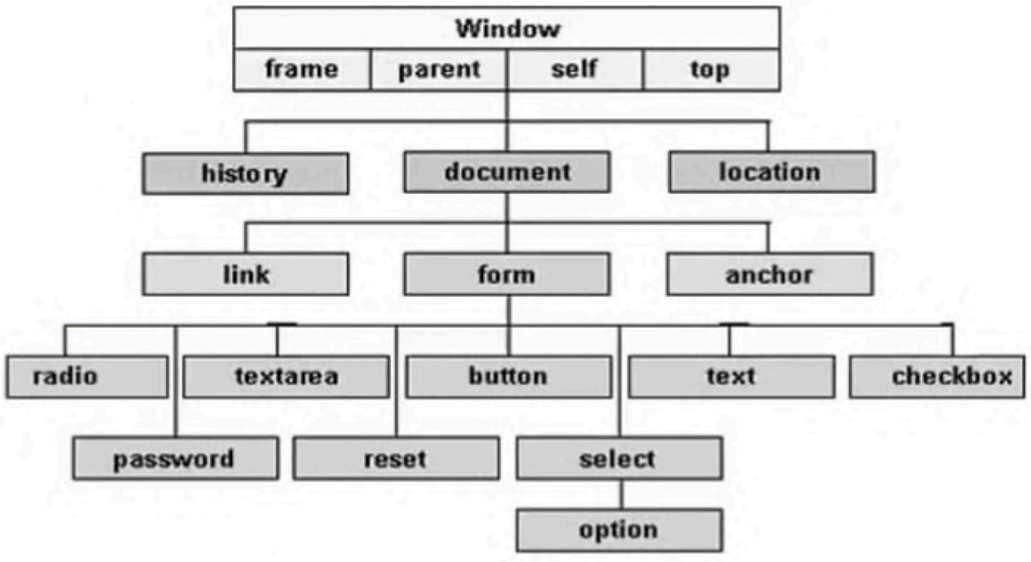



Splice is used to remove

## JavaScript - Document Object Model or DOM

- The way a document content is accessed and modified is called the Document Object Model, or DOM
- **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.

## JavaScript - Document Object Model or DOM




The whole html tree can be accessed inside the java script by the object names document

## JavaScript – Events & Event Listeners

**JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.**

```
<script type="text/javascript">
    <!--
      function sayHello() {
        alert("Hello World")
      }
    //-->
  </script>

<body>
  <p>Click the following button and see result</p>

  <form>
    <input type="button" onclick="sayHello()" value="Say Hello" />
  </form>

</body>
```

Event Listeners

Events

## JavaScript - Document Object Model or DOM

**DOM Methods**

document.getElementById("id");

document.getElementsByTagName("input");

docuemt.getElementById("divid").innerHTML="Messages";

docuemt.getElementById("divid").value

```
<head>
    <script type="text/javascript">
      //Event Listener
      function test(){
       document.getElementById("test").innerHTML="This is an inner html";
      }


    </script>

</head>
<body>

    <button onclick="test()">Click</button>

    <div id="test">


    </div>



</body>
```

```
<head>
    <script type="text/javascript">
       //Event Listener
       function test(){

        let num1 = Number.parseInt(document.getElementById("num1").value);
        let num2 = Number.parseInt(document.getElementById("num2").value);

         let sum = num1+num2;

        document.getElementById("test").innerHTML="The sum of two Numbers is "+sum;
       }


    </script>

</head>
<body>

    Enter the First Num : <input type="text" id="num1"> <br>

    Enter the Second Num : <input type="text" id="num2"> <br>

    <button onclick="test()">Add</button>
```

# SQL - Constraints

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK
- DEFAULT

## Data Types

| Data Type | Description |
|---|---|
| VARCHAR2(*size*) | Variable-length character data |
| CHAR(*size*) | Fixed-length character data |
| NUMBER(*p*,*s*) | Variable-length numeric data |
| DATE | Date and time values |
| LONG | Variable-length character data (up to 2 GB) |
| CLOB | Character data (up to 4 GB) |
| RAW and LONG RAW | Raw binary data |
| BLOB | Binary data (up to 4 GB) |
| BFILE | Binary data stored in an external file (up to 4 GB) |
| ROWID | A base-64 number system representing the unique address of a row in its table |

## SQL - DDL,Table creation (simple)

- CREATE TABLE table_name
  (column_name_1 data type,
  column_name_to data type,
  column_name_n data type)

- Other than the data types more attributes can be added to each columns of the table which are called as constraints (later).

## SQL - Create a simple table

```
CREATE TABLE dept(dno NUMBER,dname
 VARCHAR(20),city VARCHAR(20));
```

```
CREATE TABLE Employee
(Empno  NUMBER(10), Name VARCHAR (25),
Desig varchar (20), Salary varchar (20),
Deptno NUMBER);
```

## SQL Constraints – PRIMARY KEY

- Primary Key is an attribute which identifies a tuple or record set uniquely
- PRIMARY KEY = NOT NULL +UNIQUE
- The difference being one cannot have more than one primary key in a table whereas there can be more than one Unique Key..

## SQL Constraints – CHECK & DEFAULT

- The CHECK constraint is used to limit or define the value range that can be placed in a column.
- The DEFAULT constraint is used to insert a default value into a column.
- The default value will be added to all new records, if no other value is specified.

## Example :CONSTRAINTS

```
CREATE TABLE dept(dno NUMBER PRIMARY
KEY,dname VARCHAR(20) UNIQUE,city
VARCHAR(20));
```

```
CREATE TABLE Employee
(Empno   NUMBER (25) PRIMARY KEY,
Name VARCHAR (12) NOT NULL,
Desig VARCHAR (25) DEFAULT 'TRAINEE',
Salary VARCHAR (25) CHECK(Salary>=10000),
Deptno NUMBER REFERENCES dept(dno));
```

## ALTER Objects

- One can add a column, drop or modify the same, and add or drop the constraints

```
ALTER TABLE Employee DROP COLUMN Name;

ALTER TABLE Employee ADD (Name varchar (25));

ALTER TABLE Employee MODIFY(Name varchar(26));

ALTER TABLE Employee ADD CONSTRAINT SCHK CHECK
  (SALARY>1000);
```

**It adds a constraint named `schk` to the existing table `Employee`.**

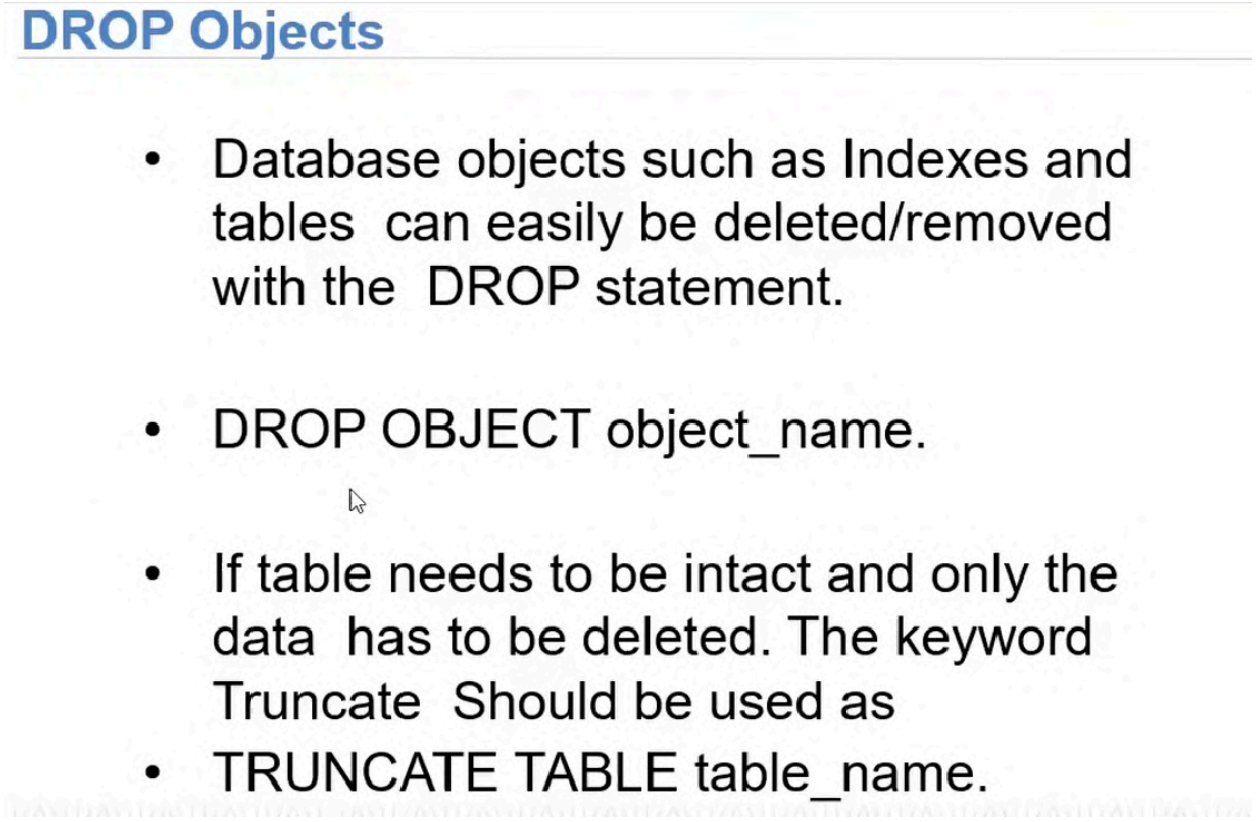
**The constraint is a CHECK constraint.**

**It ensures that any value in the existing `Salary` column must be ≥ 30000 from now on.**

**It does not change existing data.**

**It does not create a new column.**

**It does not insert or modify values like setting anything to `NULL`.**



# SQL - DML

- INSERT
- UPDATE
- DELETE

- DQL

  SELECT

## SQL – DML : INSERT

- The Insert Query will work in two ways.

1.Customized Insert – Only columns we need can be inserted leaving out others.

2.Natural Insert – All the columns of the table will be filled while inserting a record set.

## SQL – DML : INSERT

```
CREATE TABLE Employee(Eno NUMBER,
Name VARCHAR(20),Desig varchar(25), Salary
varchar (25),Deptno NUMBER);
```

```
INSERT INTO Employee(Empno,Name,Desig)
VALUES(1002,'Rajesh','Manager')
```

```
INSERT INTO Employee
VALUES(1002,'Rajesh','Manager',30000,30);
```

## SQL – DML : UPDATE

- The UPDATE statement is used to update existing records in a table.
- UPDATE table_name SET column1=value, column2=value2,... WHERE some_column=some_value;
- The WHERE clause specifies which records are be updated. If you omit the WHERE clause, all records will be updated!.

## SQL – DML : UPDATE

```
UPDATE Employee set
salary=salary+1000;
```

```
UPDATE Employee set
salary=salary+1000 WHERE deptno=20;
```

# SQL – DML : DELETE

```
• DELETE FROM Employee;

• DELETE Employee;

• DELETE FROM Employee where
  deptno=10;
```

# Basic structure of a SQLquery

| | |
|---|---|
| **General Structure** | SELECT, ALL / DISTINCT, *, AS, FROM, WHERE |
| **Comparison** | IN, BETWEEN, LIKE "% _" |
| **Grouping** | GROUP BY, HAVING,<br>COUNT( ), SUM( ), AVG( ), MAX( ), MIN( ) |
| **Display Order** | ORDER BY, ASC / DESC |
| **Logical Operators** | AND, OR, NOT |

## Simple Select

```
• SELECT * FROM Employee;
• SELECT * FROM Employee WHERE deptno=10;
• SELECT DISTINCT job FROM Employee;
• SELECT * FROM Employee WHERE deptno=10 AND
  desig='Developer';
• SELECT * FROM Employee WHERE deptno=10 OR
  desig='Developer';
• SELECT * FROM Employee WHERE Salary
  BETWEEN 20000 AND 80000;
```

## Simple Select

```
• SELECT * FROM Employee WHERE
  deptno IN (20,40);
• SELECT * FROM Employee WHERE Name
  LIKE 'S%';
• SELECT * FROM Employee WHERE Desig
  LIKE '_T%';
```

**% – Wildcard for zero or more characters**

**_ – Wildcard for a single character**

**WHERE name LIKE 'A%' -- Matches any name starting with 'A'**

**WHERE name LIKE 'A___' -- Matches any 4-letter name starting with 'A'**

## SQL FUNCTIONS

- SCALAR or SINGLE ROW FUNCTIONS
- AGGREGATE or MULTIROW FUNCTIONS

### SCALAR

```
• SELECT UPPER(Name) FROM Employee;
• SELECT * FROM Employee WHERE
  LOWER(Desig)='developer';
• SELECT SQRT(25) FROM DUAL;
• SELECT ROUND(22.67845) FROM DUAL;
```

## SQL FUNCTIONS

### AGGREGATE or MULTIROW FUNCTIONS

```
• SELECT SUM(Salary) FROM Employee;
• SELECT AVG(Salary) FROM Employee;
• SELECT MAX(Salary) FROM Employee;
• SELECT MIN(Salary) FROM Employee;
• SELECT COUNT(Salary) FROM Employee;
• SELECT COUNT(*) FROM Employee;
```

## Select – GROUP BY

SELECT ..... FROM ..... WHERE *condition*
GROUP BY *groupexpr* [HAVING *requirement*]

- **groupexpr** specifies the related rows to be grouped as one entry. Usually it is a column.

- **WHERE** condition specifies the condition of individual rows before the rows are group.

- **HAVING** requirement specifies the condition involving the whole group.

```
1
2  --SELECT Deptno,SUM(Salary) FROM Employee GROUP BY Deptno ORDER BY DEPTNO;
3  SELECT Deptno, COUNT(*) from Employee WHERE Desig!='Developer' GROUP BY Deptno HAVING COUNT (*) > 1 ORDER BY Deptno;
```

## Select – GROUP BY with HAVING CLAUSE

```
SELECT Deptno, COUNT(*) from Employee
WHERE Desig!='Developer' GROUP BY
Deptno HAVING COUNT(*) > 1 ORDER BY
Deptno;
```

## Hierarchy of Evaluation

- SELECT->WHERE->GROUP BY->GROUP FUNCTION->HAVING->ORDER BY

# Joins

- EQUI JOIN
- NON EQUI JOIN
- INNER JOIN
- OUTER JOIN (LEFT/RIGHT/FULL)
- SELF JOIN

| Type | Uses | Match Type | Example Operator |
|---|---|---|---|
| Equi Join | `=` | Exact match | `=` |
| Non-Equi Join | `!=, <, >, BETWEEN` | Range or condition | `<, >, BETWEEN` |

```
SELECT *
FROM Employees e
JOIN Departments d
ON e.DeptID = d.DeptID;
```

```
SELECT *
FROM Orders o
JOIN Discounts d
ON o.Amount BETWEEN d.MinAmount AND d.MaxAmount;
```

# Other Database Objects

| Object | Description |
|---|---|
| Table | Basic unit of storage; composed of rows |
| View | Logically represents subsets of data from one or more tables |
| Sequence | Generates numeric values |
| Index | Improves the performance of data retrieval queries |
| Synonym | Gives alternative names to objects |

## Structure of a computer system

- Hardware-CPU,Memory,I/O devices
- Operating system-
- Application programs-word processors,browsers
- Users-people,machines,other computers

# Responsibilities of OS

- Program execution
- I/O operations
- File system manipulation
- Communication
- Error detection

## Types of OS

- Single User Single Tasking OS- eg : MS-DOS
- Single User Multitasking OS – eg: Windows
- Multi user Multiprogramming OS- eg : Unix

# Functions of OS

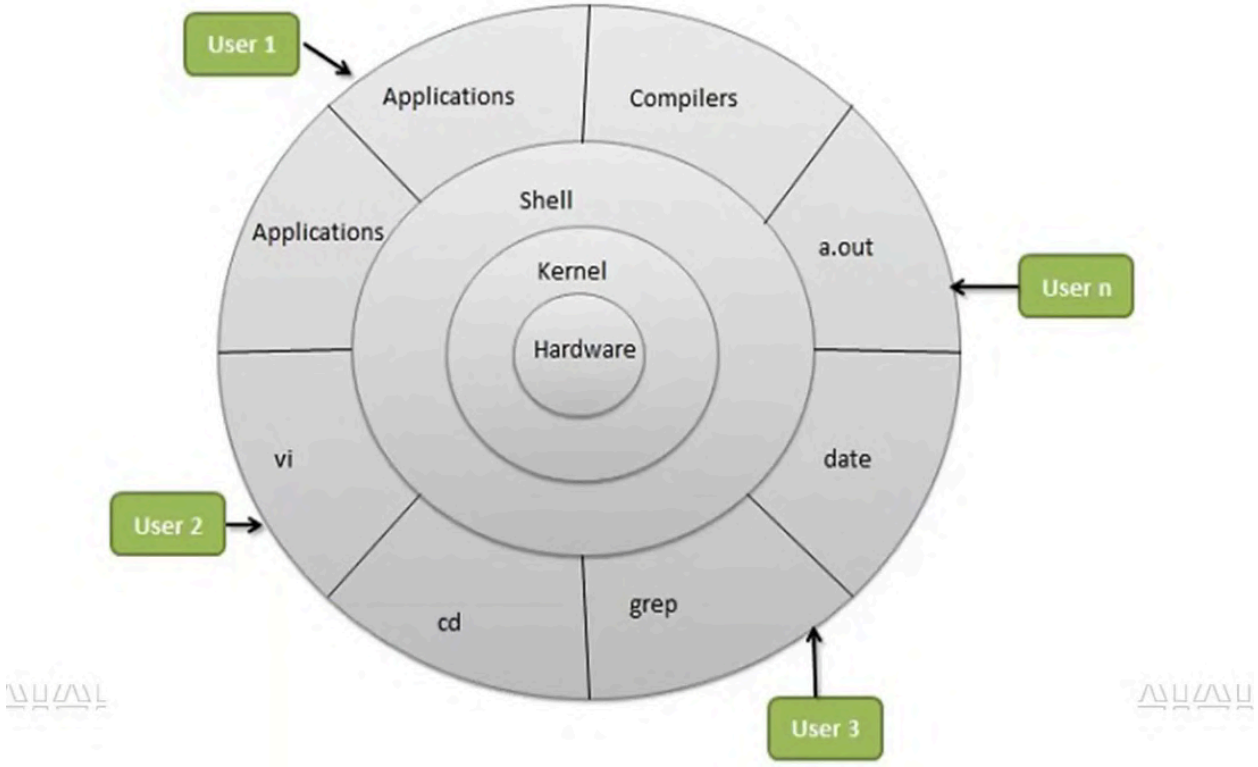
- Process management
- Memory management
- Storage management
- Device management

## Unix-introduction

- Unix was originally developed in 1969 by a group of AT&T employees Ken Thompson, Dennis Ritchie, Douglas McIlroy, and Joe Ossanna at Bell Labs.

- There are various Unix variants available in the market. Solaris Unix, AIX, HP Unix and BSD are a few examples.
Linux is also a flavor of Unix which is freely available.

## Unix Architecture




# 1. Hardware Layer (Bottom Layer)

- This is the **physical system**: CPU, memory (RAM), hard disk, network cards, etc.
- The **kernel** interacts directly with this hardware.
- **Example**: Keyboard, monitor, hard drive, mouse, etc.

# 2. Kernel Layer (The Heart of UNIX)

The **kernel** is the **core** of the UNIX operating system. It manages everything happening inside the system.

**What the kernel does:**

| Task | Explanation |
|---|---|
| **Process Management** | Handles starting, stopping, scheduling processes |
| **Memory Management** | Allocates and deallocates RAM to processes |
| **File System Management** | Manages files, directories, and permissions |
| **Device Management** | Communicates with hardware (keyboard, disk, printer) |
| **System Calls Handling** | Provides services to programs (via system calls) |

It stays in memory always and is not directly accessed by users.

## 3. Shell Layer (The Interface )

The **shell** is a **command-line interpreter** — it takes commands from the user and tells the kernel to execute them.

### Types of shells:

| Shell | Description |
|---|---|
| `sh` | Bourne Shell |
| `bash` | Bourne Again Shell (most common in Linux) |
| `csh` | C Shell |
| `ksh` | Korn Shell |

### What shell does:

- Takes **user input**
- Interprets it
- Sends it to the **kernel**
- Displays the result

It acts like a **translator** between the user and the kernel.

## 4. Application Layer (User Programs )

This is where **you**, the user, interact with UNIX.

### Examples:

| Program | Use |
|---|---|
| `ls` | List files |
| `nano`, `vi` | Text editors |
| `cp`, `mv`, `rm` | File operations |
| `gcc` | Compiler for C/C++ |

> These programs **rely on the shell** and **use system calls** to request services from the kernel.

## How Everything Works Together (Example Flow)

Let's say you type:

ls -l

1. **You** enter the command in the terminal (application layer).
2. The **shell** interprets it and tells the **kernel**: “Run the `ls` program with the `-l` option.”
3. The **kernel**:
    - Loads the `ls` program into memory
    - Reads the files from the hard disk
    - Sends the output back
4. The **shell** displays the result to **you**.

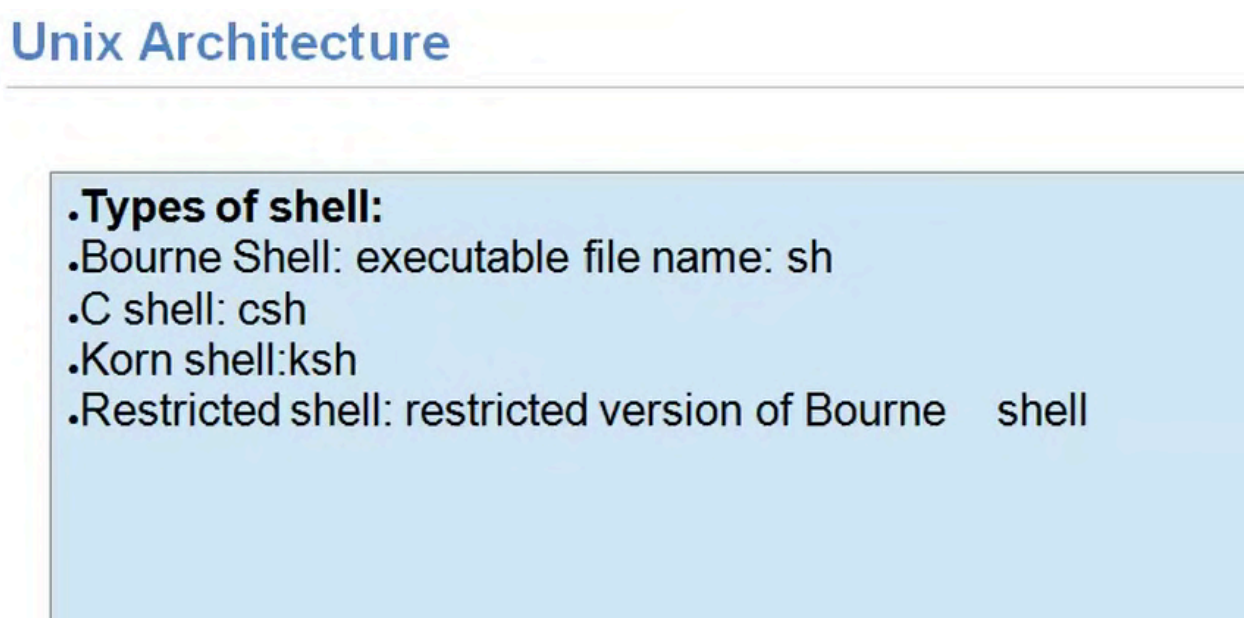
## Simple Analogy

Think of UNIX like a restaurant:

| Layer | Role in Restaurant 🍽️ |
|---|---|
| Hardware | Kitchen, stove, fridge (physical) |
| Kernel | Chef — manages cooking, orders |
| Shell | Waiter — takes your order |
| Application | You — the customer |

## Key Benefits of UNIX Architecture

- **Modular**: Each part (kernel, shell, apps) is separate and manageable
- **Stable & Secure**
- **Portable**: Can run on many hardware types
- **Multi-user** and **multi-tasking**



## Getting started with Ubuntu

→The "Ubuntu" is an ancient Zulu and Xhosa word which means "humanity to others"

→Ubuntu is a free Operating System which can run on PC or laptop instead of Windows or OSX

→It is safe, hugely powerful, and fun

→The word 'Ubuntu' has its origin in the Bantu languages of South Africa

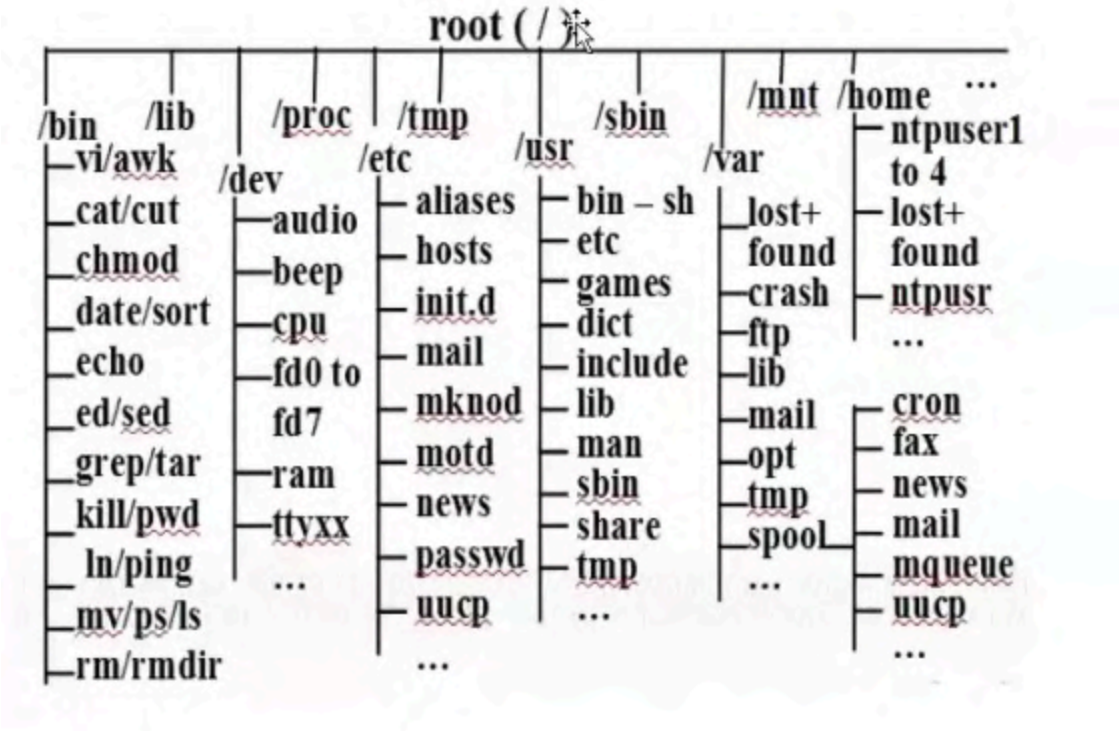
→Ubuntu Linux is one of the flavour of the Linux Operating System

## Introduction to File System

- A UNIX file is featureless because it is simply an array of bytes.
- Dominant file type in UNIX is the text file.
- System related files are also stored in text form.
- Separate device can be added by creating a file for it.
- Root is the supremo and is represented by the '/'. Every subdirectory must have a parent.
- File names can be up to 14 characters long; can contain both upper and lower case alphabets, digits, a dot, hyphen (-), underscore (_) anywhere; should not have a blank or tab; are case-sensitive.
- Path names are a sequence of directory names separated by '/'. They are used to access files.

Absolute pathname - file location is determined with respect to the root.

Relative pathname - file location is determined with respect to the current directory.




# 1. File and Directory Commands

| Command | Description |
|---|---|
| `ls` | Lists files and directories |
| `ls -l` | Lists in long format (with permissions, size, owner, etc.) |
| `ls -a` | Shows hidden files (starting with `.`) |
| `pwd` | Prints the current working directory |
| `cd` | Changes directory |
| `cd ..` | Moves up one directory level |
| `mkdir` | Creates a new directory |
| `rmdir` | Deletes an empty directory |
| `rm -r` | Deletes a directory and its contents recursively |

| | |
|---|---|
| `touch` | Creates a new empty file |
| `cp` | Copies files or directories |
| `mv` | Moves or renames files/directories |
| `rm` | Removes/deletes files |
| `find` | Finds files and directories based on conditions |
| `locate` | Quickly finds files (uses a database; faster than `find`) |

## 2. File Viewing and Editing

| Command | Description |
|---|---|
| `cat` | Displays content of a file |
| `more` | Views file content page by page (scrollable) |
| `less` | Like `more`, but more powerful (scroll up/down) |
| `head` | Shows the first 10 lines of a file |
| `tail` | Shows the last 10 lines of a file |
| `tail -f` | Monitors a file in real time (used for logs) |
| `nano` | Simple terminal text editor |
| `vi` or `vim` | Powerful terminal-based text editor |

## 3. File Permissions and Ownership

| Command | Description |
|---|---|
| `chmod` | Changes file permissions |
| `chown` | Changes file owner |
| `chgrp` | Changes group ownership of a file |

| | |
|---|---|
| `umask` | Sets default file permissions when files are created |

## 4. Process Management

| Command | Description |
|---|---|
| `ps` | Lists running processes |
| `top` | Live display of running processes and system usage |
| `kill` | Terminates a process by PID |
| `killall` | Kills processes by name |
| `nice` | Starts a process with a given priority |
| `renice` | Changes priority of an existing process |
| `bg` | Moves a job to the background |
| `fg` | Brings a background job to foreground |
| `jobs` | Lists background jobs |

## 5. Disk and System Info

| Command | Description |
|---|---|
| `df` | Shows disk space usage |
| `du` | Shows directory/file sizes |
| `free` | Shows memory usage |
| `uname -a` | Shows system information |
| `uptime` | Shows how long the system has been running |
| `whoami` | Displays current logged-in user |
| `hostname` | Shows or sets the system hostname |

## 6. Networking Commands

| Command | Description |
|---|---|
| `ping` | Checks connectivity to another host |
| `ifconfig` / `ip` | Displays network interfaces info |
| `netstat` | Shows network connections (deprecated, use `ss`) |
| `ss` | Shows socket statistics (modern netstat replacement) |
| `curl` | Transfers data from/to a server (HTTP requests, etc.) |
| `wget` | Downloads files from the web |
| `scp` | Securely copies files between hosts |
| `ssh` | Connects to another machine securely |
| `ftp` / `sftp` | Transfers files using FTP/SFTP |

## 7. Package Management (varies by OS)

| Command | Description |
|---|---|
| `apt` / `apt-get` | Package manager on Debian/Ubuntu |
| `yum` / `dnf` | Package manager on Red Hat/CentOS/Fedora |
| `pacman` | Package manager on Arch Linux |
| `zypper` | Package manager on openSUSE |

## 8. Compression and Archiving

| Command | Description |
|---|---|

| | |
|---|---|
| `tar` | Archives multiple files into one (`.tar`) |
| `tar -czf` | Creates compressed archive (`.tar.gz`) |
| `gzip` | Compresses files |
| `gunzip` | Decompresses `.gz` files |
| `zip` | Creates `.zip` files |
| `unzip` | Extracts `.zip` files |

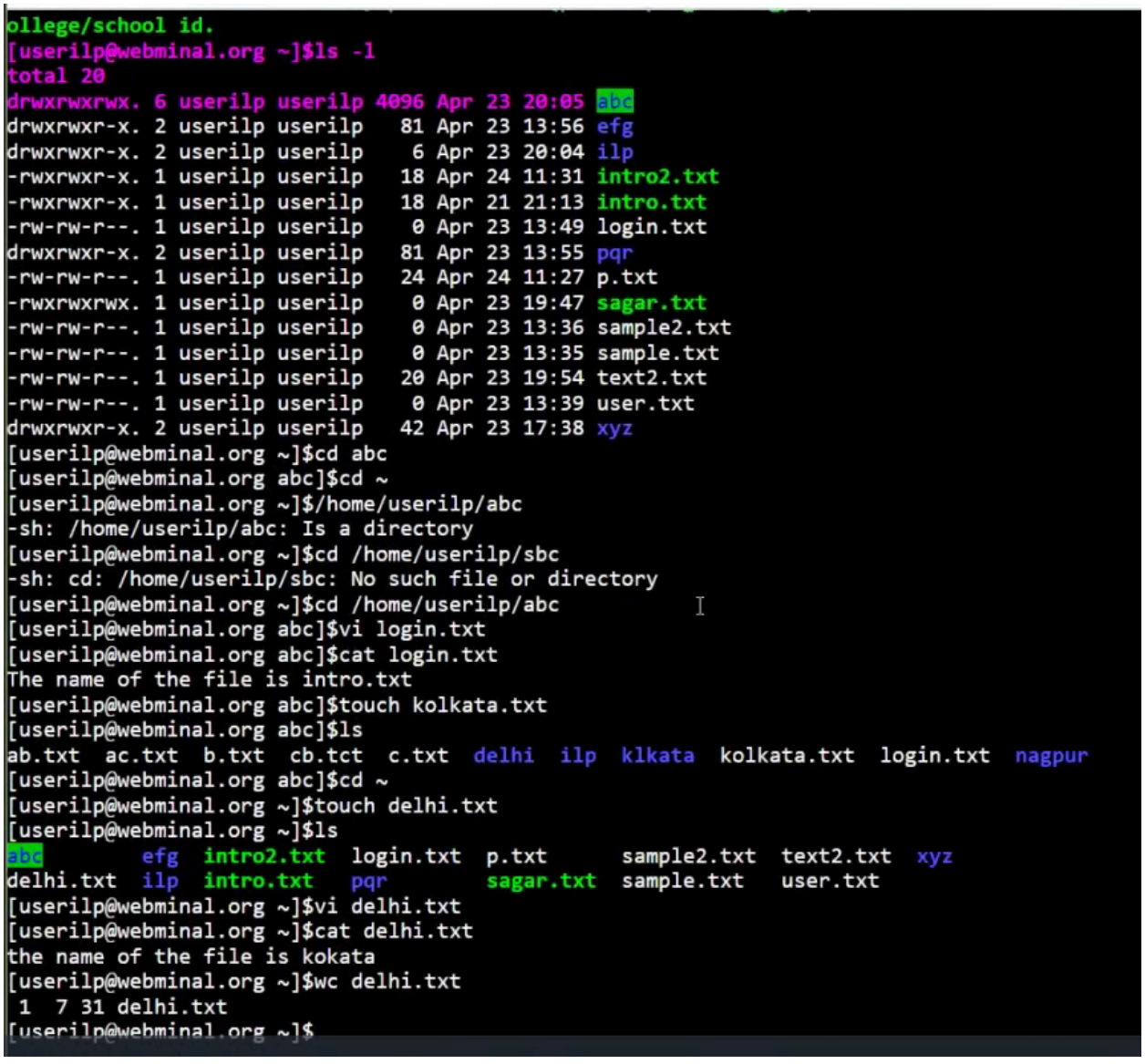
## 9. User Management (needs root privileges)

| Command | Description |
|---|---|
| `adduser` / `useradd` | Adds a new user |
| `passwd` | Changes user password |
| `deluser` / `userdel` | Deletes a user |
| `su` | Switches user (default to root) |
| `sudo` | Runs a command as another user (usually root) |

## 10. Miscellaneous

| Command | Description |
|---|---|
| `history` | Shows command history |
| `alias` | Creates a shortcut for a command |
| `echo` | Prints text to the terminal |
| `date` | Displays current date and time |
| `cal` | Shows calendar |

| | |
|---|---|
| `man` | Displays manual/help for commands |
| `clear` | Clears the terminal |
| `exit` | Exits the terminal or script |



```
[userilp@webminal.org ~]$wc delhi.txt
 1  7 31 delhi.txt
[userilp@webminal.org ~]$cp p.txt q.txt
[userilp@webminal.org ~]$ls
abc         efg  intro2.txt  login.txt  p.txt  sagar.txt    sample.txt  user.txt
delhi.txt   ilp  intro.txt   pqr        q.txt  sample2.txt  text2.txt   xyz
[userilp@webminal.org ~]$mv delhi.txt mumbai.txt
[userilp@webminal.org ~]$ls
abc  ilp         intro.txt  mumbai.txt  p.txt  sagar.txt    sample.txt  user.txt
efg  intro2.txt  login.txt  pqr         q.txt  sample2.txt  text2.txt   xyz
[userilp@webminal.org ~]$rm mumbai.txt
[userilp@webminal.org ~]$ls
abc  ilp         intro.txt  pqr    q.txt      sample2.txt  text2.txt  xyz
efg  intro2.txt  login.txt  p.txt  sagar.txt  sample.txt   user.txt
[userilp@webminal.org ~]$ls
abc  ilp         intro.txt  pqr    q.txt      sample2.txt  text2.txt  xyz
efg  intro2.txt  login.txt  p.txt  sagar.txt  sample.txt   user.txt
[userilp@webminal.org ~]$ls -l
total 24
drwxrwxrwx. 6 userilp userilp 4096 Apr 24 12:45 abc
drwxrwxr-x. 2 userilp userilp   81 Apr 23 13:56 efg
drwxrwxr-x. 2 userilp userilp    6 Apr 23 20:04 ilp
-rwxrwxr-x. 1 userilp userilp   18 Apr 24 11:31 intro2.txt
-rwxrwxr-x. 1 userilp userilp   18 Apr 21 21:13 intro.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:49 login.txt
drwxrwxr-x. 2 userilp userilp   81 Apr 23 13:55 pqr
-rw-rw-r--. 1 userilp userilp   24 Apr 24 11:27 p.txt
-rw-rw-r--. 1 userilp userilp   24 Apr 24 12:50 q.txt
-rwxrwxrwx. 1 userilp userilp    0 Apr 23 19:47 sagar.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:36 sample2.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:35 sample.txt
-rw-rw-r--. 1 userilp userilp   20 Apr 23 19:54 text2.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:39 user.txt
drwxrwxr-x. 2 userilp userilp   42 Apr 23 17:38 xyz
[userilp@webminal.org ~]$
```

## Directories

- Directories store both special and ordinary files.
- For users familiar with Windows or Mac OS, UNIX directories are equivalent to folders.
- A directory file contains an entry for every file and subdirectory that it houses.
- Never contain "real" information which you would work with (such as text). Basically, just used for organizing files.
- All files are descendants of the root directory, ( named / ) located at the top of the tree.
- In long-format output of ls –l , this type of file is specified by the "d" symbol.

## login

**There are three commands available to get you this information, based on how much you wish to know about the other users: users, who, and w.**

```
$ users
 amar bablu qadir

$ who
amrood ttyp0 Oct 8 14:10 (limbo)
bablu  ttyp2 Oct 4 09:08 (calliope)
qadir  ttyp4 Oct 8 12:09 (dent)

$
```

Try the **w** command on your system to check the output. This lists down information associated with the users logged in the system.

## Types of Permissions

**The permissions of a file are the first line of defense in the security of a Unix system.**

**Access to a file has three levels:**

Read

Grants the capability to read, i.e., view the contents of the file.

Write

Grants the capability to modify, or remove the content of the file.

Execute

User with execute permissions can run a file as a program.

**Each file is associated with a set of identifiers that are used to determine who can access the file:**

**User ID (UID)** – Specifies the user that owns the file. By default, this is the creator of the file.

**Group ID (GID)** – Specifies the user-group that the file belongs to.

**Finally, there are three sets of access permissions associated with each file:**

**User permission –** Specifies the level of access given to the user matching the file's UID.

**Group permissions** − The group's permissions determine what actions a user, who is a member of the group that a file belongs to, can perform on the file.

**Other (world) permissions** − The permissions for others indicate what action all other users can perform on the file.

Together, this scheme of access controls makes the Unix system extremely secure while simultaneously providing the flexibility required of a multi-user system.

```
[userilp@webminal.org ~]$cd abc
[userilp@webminal.org abc]$cd ~
[userilp@webminal.org ~]$cd abc
[userilp@webminal.org abc]$pwd
/home/userilp/abc
[userilp@webminal.org abc]$cd ~
[userilp@webminal.org ~]$mkdir batch
[userilp@webminal.org ~]$ls
abc    efg  intro2.txt  login.txt  p.txt  sagar.txt    sample.txt  user.txt
batch  ilp  intro.txt   pqr        q.txt  sample2.txt  text2.txt   xyz
[userilp@webminal.org ~]$rmdir abc
rmdir: failed to remove ‘abc’: Directory not empty
[userilp@webminal.org ~]$rmdir batch
[userilp@webminal.org ~]$ls
abc  ilp         intro.txt  pqr    q.txt      sample2.txt  text2.txt  xyz
efg  intro2.txt  login.txt  p.txt  sagar.txt  sample.txt   user.txt
[userilp@webminal.org ~]$ls -l
-sh: -l: command not found
[userilp@webminal.org ~]$ls -l
total 24
drwxrwxrwx. 6 userilp userilp 4096 Apr 24 12:45 abc
drwxrwxr-x. 2 userilp userilp   81 Apr 23 13:56 efg
drwxrwxr-x. 2 userilp userilp    6 Apr 23 20:04 ilp
-rwxrwxr-x. 1 userilp userilp   18 Apr 24 11:31 intro2.txt
-rwxrwxr-x. 1 userilp userilp   18 Apr 21 21:13 intro.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:49 login.txt
drwxrwxr-x. 2 userilp userilp   81 Apr 23 13:55 pqr
-rw-rw-r--. 1 userilp userilp   24 Apr 24 11:27 p.txt
-rw-rw-r--. 1 userilp userilp   24 Apr 24 12:50 q.txt
-rwxrwxrwx. 1 userilp userilp    0 Apr 23 19:47 sagar.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:36 sample2.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:35 sample.txt
-rw-rw-r--. 1 userilp userilp   20 Apr 23 19:54 text2.txt
-rw-rw-r--. 1 userilp userilp    0 Apr 23 13:39 user.txt
drwxrwxr-x. 2 userilp userilp   42 Apr 23 17:38 xyz
[userilp@webminal.org ~]$
```

[The initial **dash** ("**-**") in the first permissions string indicates the entry is a file. The letter "**d**" instead of a ("**-**") in the 2nd example indicates the entry is a directory.]

The **first three positions** (after the "**-**" or "**d**") designate owner's permissions. The **r** indicates the owner can **r**ead the file. The **w** indicates the owner can **w**rite to the file. The **x** indicates the owner can e**x**ecute the file.

The **second three positions** designate permissions for the group. In this example, the group members can **r**ead, but **not w**rite to it or e**x**ecute it. (Execution permission is usually only given for particular files or a specific directory.) *If you don't give others access to write or execute files, ignore this part.*

The **last three positions** are for the world/anyone. To allow your Web pages to be viewed using a browser, you need this permission set to "**r**ead."

The **number** before the user name indicates the levels of directories.

**Username** indicates the owner of the file. *Your username appears in this position.*

User name k baad groupname , aur abhi qki koi group nahi hai to vo same as username aajata hai

## Setting Permissions

### 1. chmod: change file access permissions

description: This command is used to change the file permissions. These permissions are read, write and execute permission for the owner, group, and others.

**syntax (symbolic mode):**

chmod [ugoa][[+-=][mode]] file

The first optional parameter indicates who – this can be (u)ser, (g)roup, (o)thers or (a)ll

The second optional parameter indicates opcode – this can be for adding (+), removing (-) or assigning (=) permission.

The third optional parameter indicates the mode – this can be (r)ead, (w)rite, or e(x)ecute.

## chown: change ownership of the file.

- description: Only the owner of the file has the rights to change the file ownership.
- syntax: chown [owner] [file]
- Example: Change the owner of file1 to user2 assuming that it is currently owned by the current user

$ chown user2 file1

The **chown** command changes the ownership of a file. The basic syntax is as follows −

$ chown user filelist

The value of the user can be either the **name of a user** on the system or the **user id (uid)** of a user on the system.

Permission examples:

chmod a+r files are readable by all
chmod a-r files cancels the ability for all to read the file
chmod a-rwx cancels all access for all
chmod g+rw files give the group read and write permission
chmod u+rwx files give the owner all permissions
chmod og+rw files give the world and the group read and write permission

a mtlb all the users

| OWNER (USER) | | | GROUP (if you share a directory.file) | | | PUBLIC (OTHERS) | | |
|---|---|---|---|---|---|---|---|---|
| R | W | X | R | W | X | R | W | X |
| 400 | 200 | 100 | 40 | 20 | 10 | 4 | 2 | 1 |

In order to translate the mode you require to a number, simply add the numbers corresponding to the individual permissions you want.

So, if you want the files to be readable and writable by the owner (that is, you), readable by the group, and readable by all the other users of the system, you perform the addition:

| | | | | |
|---|---|---|---|---|
| 4 | 0 | 0 | owner | **R** ead |
| 2 | 0 | 0 | owner | **W** rite |
| 1 | 0 | 0 | owner | **X** ecute |
| , | 4 | 0 | group | **R** ead |
| , | 1 | 0 | group | **X** ecute |
| , | , | 4 | public | **R** ead |
| , | , | 1 | public | **X** ecute |
| – | – | – | | |
| 7 | 5 | 5 | | |

Your chmod permission command will look like:

chmod 755 (Leave a space and type in filename.)

Owner can read, write, execute files

Group can read and execute (use) but not change files.

Public can read and execute (use) but not change.

**Examples:** Observe the following sequence of command execution statements and analyze the output

1. Let us view the initial permissions of the file ***abc***

   ls -l ***abc***

   -rw-r--r-- 1 735873 oinstall 0 Feb 7 12:37 abc

If we observe the output, initially owner have read and write permissions and group and others have only read permission

2. Let us remove all the permission from all users for ***abc*** file, as below

   ***chmod 0 abc***

   Let us verify the changed permissions

   ls -l abc

   ---------- 1 735873 oinstall 0 Feb 7 12:37 abc

   Yes all permissions are taken away from all users on the file abc

3. To provide all permissions to the user, read permission to group and write and execute permissions for Others,

   use the below command

   ***chmod 743 abc***

Let us verify the changed permissions

ls -l abc

-rwxr---wx 1 735873 oinstall 0 Feb 7 12:37 abc

From the output all permissions to user, read permission to group and write and execute

permissions for Others, Use the below command

4. ***chmod 755 abc***

   The above command provides all permissions to the user, read and execute permissions to group and others

   Let us verify ls -l abc

   -rwxr-xr-x 1 735873 oinstall 0 Feb 7 12:37 abc

5. ***chmod 777 abc***

   The above command provides all permissions to all users . We can verify using below command

   ls -l abc

   -rwxrwxrwx 1 735873 oinstall 0 Feb 7 12:37 abc

Environment variables are **global values** that are **accessible system-wide** to influence the behavior of **processes**, **commands**, and **applications**.

## Think of them like:

> **"Settings" or "shortcuts" that tell the system how to behave.**

## Common Examples:

| Variable | Meaning |
|---|---|
| `PATH` | Tells the system where to look for executables |
| `HOME` | Path to the current user's home directory |
| `USER` | Username of the currently logged-in user |
| `SHELL` | Default shell (e.g., `/bin/bash`) |
| `PWD` | Current working directory |
| `EDITOR` | Default text editor (e.g., `nano`, `vim`) |
| `LANG` | Language/locale settings |

## Example:

```
echo $HOME
```

Output:

```
/home/anshika
```

```
echo $PATH
```

Shows directories where system looks for commands.

**What is a Computing Environment?**

The Computing environment is the Platform(Platform = Operating System+ Processor) where a user can run programs.

**What is a Variable?**

In computer science, a **variable is a location for storing a value which can be a filename**, **text**, **number** or any other **data**.

It is usually referred to with its Symbolic name which is given to it while creation. The value thus stored can be displayed, deleted, edited and re-saved.

Variables play an important role in computer programming because they enable programmers to write flexible programs. As they are related to the Operating system that we work on

Make sure that you type the variable name in the right letter case otherwise you may not get the desired results.

The **'env'** command displays **all the environment variables.**

You can create your own user defined variable, with syntax

VARIABLE_NAME= variable_value

Again, bear in mind that variables are case-sensitive and usually they are created in upper case.

If you want to access the variable just use $ symbol in front of it .

The following syntax can be used to remove a Variable from the system.

unset variablename

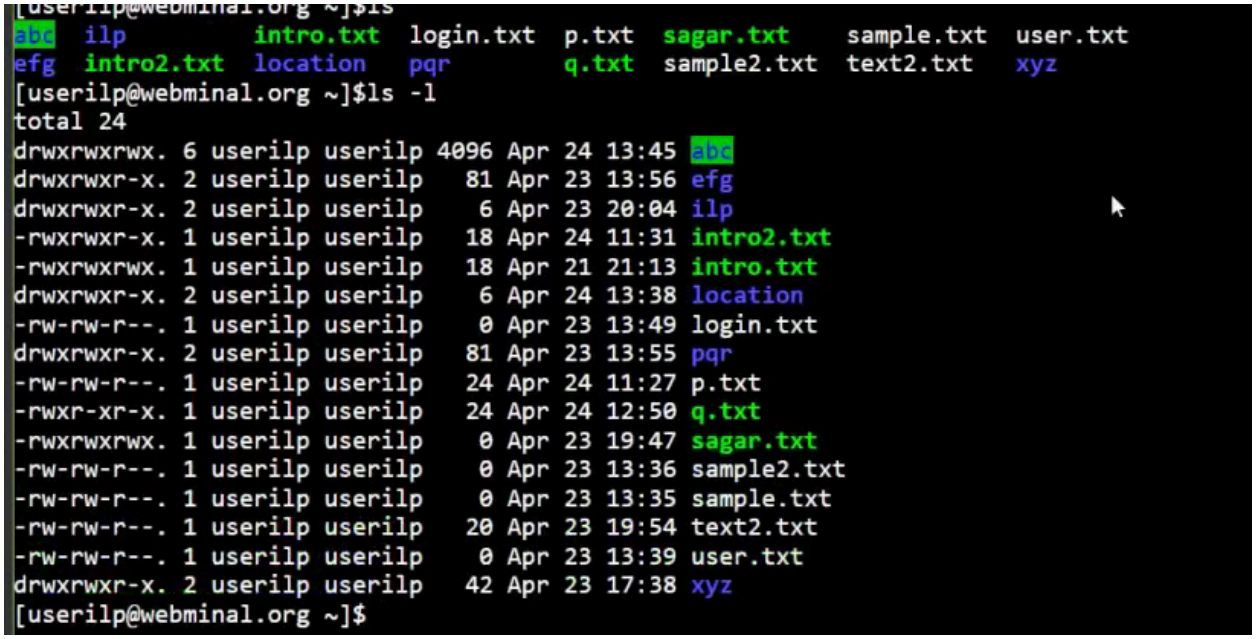
This **would remove the Variable** and its value permanently.

Below are some of the most common environment variables:

USER - The current logged in user.
HOME - The home directory of the current user.
SHELL - The path of the current user's shell, such as bash or zsh.
LOGNAME - The name of the current user.
PATH - A list of directories to be searched when executing commands. When you run a command the system will search those directories in this order and use the first found executable.
LANG - The current locales settings.
TERM - The current terminal emulation.
MAIL - Location of where the current user's mail is stored.

echo ki jgh printenv bhi use kiya jaa skta hai

echo k sath $ chahiye , printenv k sath nahi




| Command | What It Shows | Details ? |
|---|---|---|
| `ls` | Just names of files and dirs | ❌ No |
| `ls -l` | Full info (permissions, size, etc.) | ✅ Yes |

You can use the **wc** command to get a count of the total number of lines, words, and characters contained in a file.

Following is a simple example to see the information about the file created above −

```
$ wc filename
2  19 103 filename
$
```

Here is the detail of all the four columns −

**First Column** − Represents the total number of lines in the file.

**Second Column** − Represents the total number of words in the file.

**Third Column** − Represents the total number of bytes in the file. This is the actual size of the file.

**Fourth Column** − Represents the file name.

You can give multiple files and get information about those files at a time. Following is simple syntax −

```
$ wc filename1 filename2 filename3
```

## File Management Commands

### ls COMMAND

The ls command lists all files in the directory that match the name. If name is left blank, it will list all of the files in the directory.

### *Meta Characters:*

Meta characters have special meaning in Unix. For example * and ? are metacharacters. We use * to match 0 or more characters, a question mark ? matches with single character.

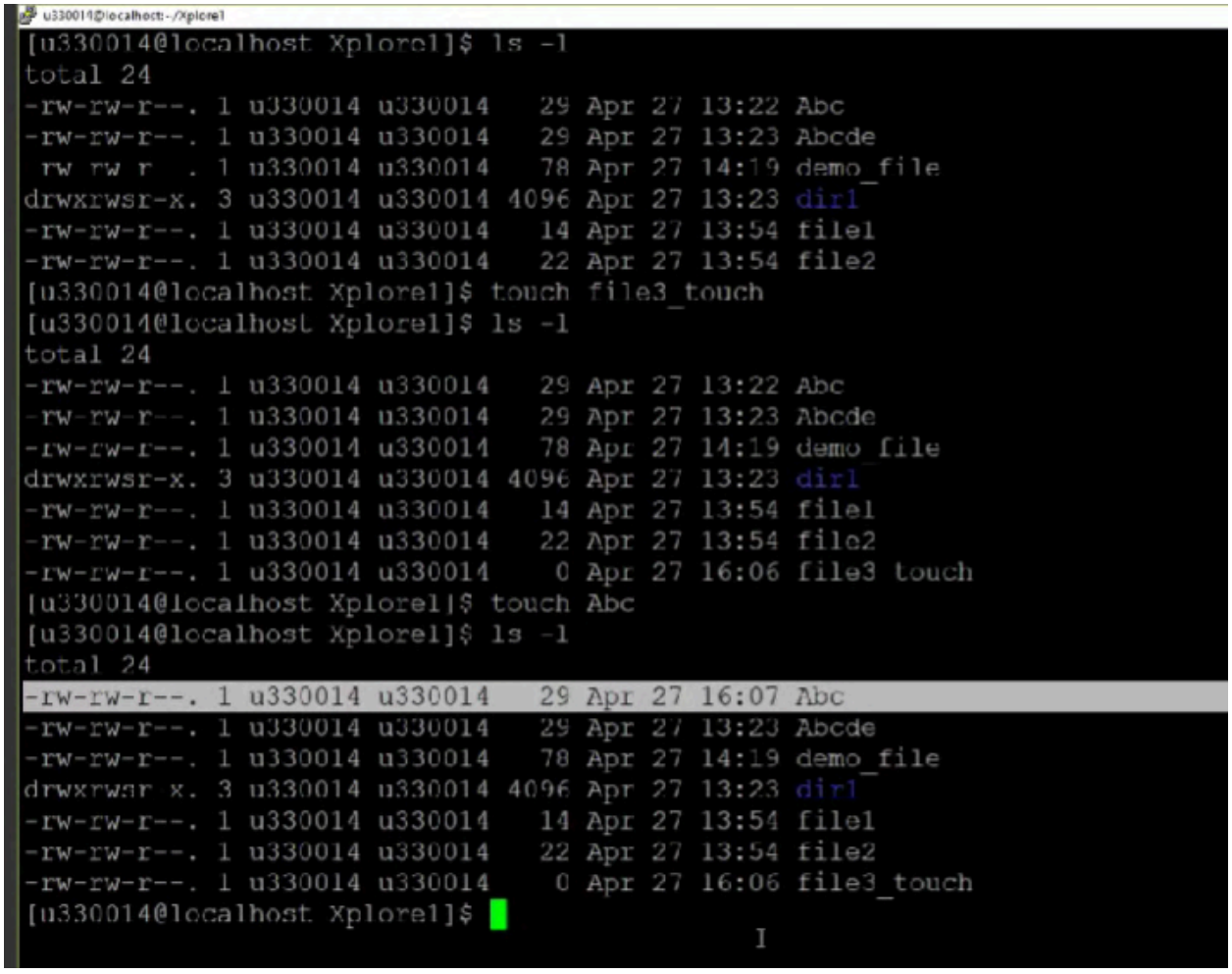

* means followed by any characters , ? means an unknown character

## Creating a File

We can create a file using Various method

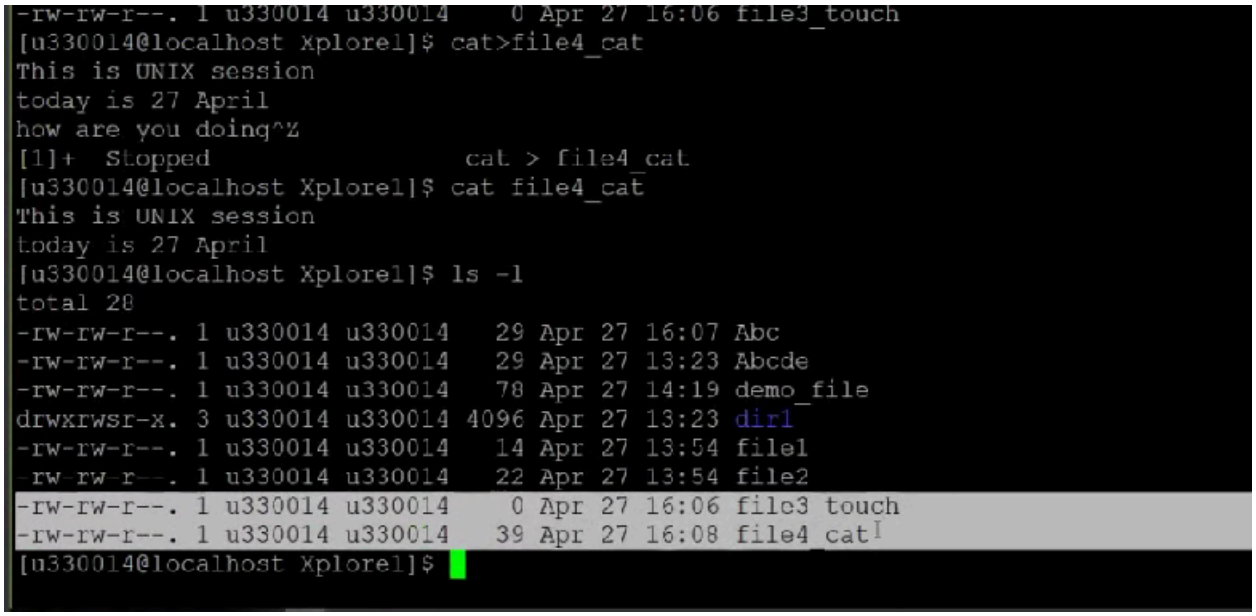
- Touch
- cat>TestFile
  content
- Using VI Editor

Use of touch command :

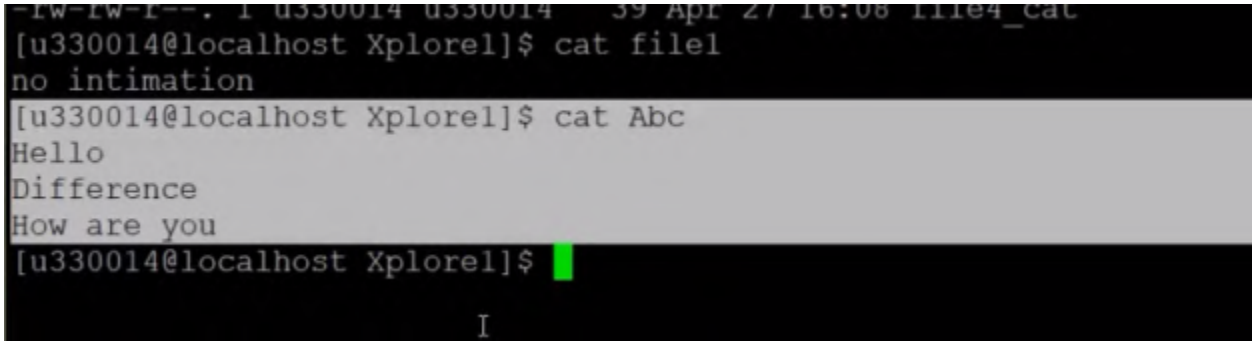



Creates new file as well as can change the timestamp of the existing file

Use of cat command




We can also write anything in the file after using the command , and to come out of the test we use ctrl+v and using cat file name then we can display the content of it



## Commands

### Displaying Content of a File

#### Cat Command

cat command can be used to see the content of a file.Example to see the content of above created file:

$ cat filename

### Counting Words in a File

#### wc Command

wc command can be used to get a count of the total number of lines, words, and characters contained in a file.

$ wc filename

```
[u330014@localhost Xplore1]$ cat Abc
Hello
Difference
How are you
[u330014@localhost Xplore1]$ wc Abc
 3  5 29 Abc
[u330014@localhost Xplore1]$ wc -w Abc
5 Abc
[u330014@localhost Xplore1]$ wc -l Abc
3 Abc
[u330014@localhost Xplore1]$ wc -c Abc
29 Abc
[u330014@localhost Xplore1]$
```

3 lines 5 words 29 characters

## Copying Files

### cp command

cp command used to copy a file

$cp file1 file2

This will copy the contents of File 1 to File 2.

## Renaming Files

### mv Command

mv command used to rename files.

mv file1 file2

This will rename the File.

```
[u330014@localhost Xplore1]$ cp Abc Abc_copy
[u330014@localhost Xplore1]$ ls -l
total 32
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 16:07 Abc
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 16:12 Abc_copy
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 13:23 Abcde
-rw-rw-r--. 1 u330014 u330014   78 Apr 27 14:19 demo_file
drwxrwsr-x. 3 u330014 u330014 4096 Apr 27 13:23 dir1
-rw-rw-r--. 1 u330014 u330014   14 Apr 27 13:54 file1
-rw-rw-r--. 1 u330014 u330014   22 Apr 27 13:54 file2
-rw-rw-r--. 1 u330014 u330014    0 Apr 27 16:06 file3_touch
-rw-rw-r--. 1 u330014 u330014   39 Apr 27 16:08 file4_cat
[u330014@localhost Xplore1]$ cat Abc_copy
Hello
Difference
How are you
```

```
[u330014@localhost Xplore1]$ mv Abc_copy Abc_move
[u330014@localhost Xplore1]$ ls -l
total 32
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 16:07 Abc
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 13:23 Abcde
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 16:12 Abc_move
-rw-rw-r--. 1 u330014 u330014   78 Apr 27 14:19 demo_file
drwxrwsr-x. 3 u330014 u330014 4096 Apr 27 13:23 dir1
-rw-rw-r--. 1 u330014 u330014   14 Apr 27 13:54 file1
-rw-rw-r--. 1 u330014 u330014   22 Apr 27 13:54 file2
-rw-rw-r--. 1 u330014 u330014    0 Apr 27 16:06 file3_touch
-rw-rw-r--. 1 u330014 u330014   39 Apr 27 16:08 file4_cat
[u330014@localhost Xplore1]$
```

# Deleting Files

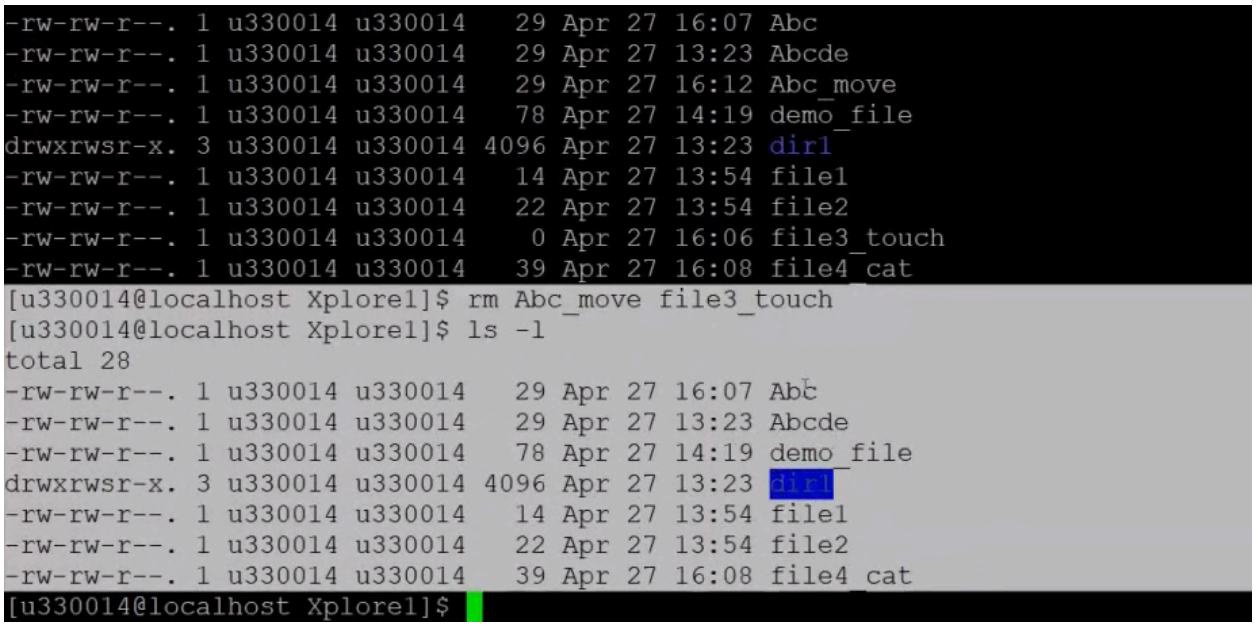
## rm Command

rm Command used to remove the files

$rm Filename

You can remove multiple files at a time by using

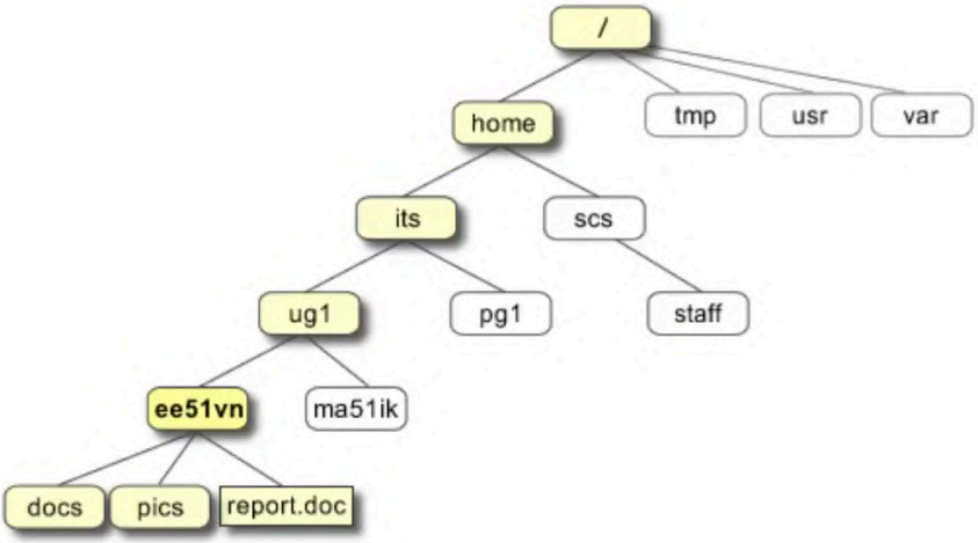
$ rm filename1 filename2 filename3




Multiple files can be removed at the same time

## Directory Structure - Diagram

A sample directory structure in UNIX is shown below




## Working with Directory

The directory in which the login was done for any user is called **home** directory.

User can go to home directory anytime using the following command:

$cd ~

~ indicates home directory.

### cd Command

cd command is used to change between Directories

cd [directory]

### cd.. Command

cd.. Command is used to go back by one directory

## Working with Directories

### mkdir command

mkdir commands is used to create directories in UNIX.

$mkdir dirname

$mkdir docs pub (Under Current Directory)

The above command creates two directories at a time with name docs and pub.

### mv command

The mv (move) command can also be used to rename a directory.

The syntax is as follows:

$mv olddir newdir

```
[u330014@localhost Xplore1]$ mkdir dir2 dir3
[u330014@localhost Xplore1]$ ls -l
total 36
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 16:07 Abc
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 13:23 Abcde
-rw-rw-r--. 1 u330014 u330014   78 Apr 27 14:19 demo_file
drwxrwsr-x. 3 u330014 u330014 4096 Apr 27 13:23 dir1
drwxrwsr-x. 2 u330014 u330014 4096 Apr 27 16:25 dir2
drwxrwsr-x. 2 u330014 u330014 4096 Apr 27 16:25 dir3
-rw-rw-r--. 1 u330014 u330014   14 Apr 27 13:54 file1
-rw-rw-r--. 1 u330014 u330014   22 Apr 27 13:54 file2
-rw-rw-r--. 1 u330014 u330014   39 Apr 27 16:08 file4_cat
[u330014@localhost Xplore1]$ mv dir2 dir4
[u330014@localhost Xplore1]$ ls -l
total 36
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 16:07 Abc
-rw-rw-r--. 1 u330014 u330014   29 Apr 27 13:23 Abcde
-rw-rw-r--. 1 u330014 u330014   78 Apr 27 14:19 demo_file
drwxrwsr-x. 3 u330014 u330014 4096 Apr 27 13:23 dir1
drwxrwsr-x. 2 u330014 u330014 4096 Apr 27 16:25 dir3
drwxrwsr-x. 2 u330014 u330014 4096 Apr 27 16:25 dir4
-rw-rw-r--. 1 u330014 u330014   14 Apr 27 13:54 file1
-rw-rw-r--. 1 u330014 u330014   22 Apr 27 13:54 file2
-rw-rw-r--. 1 u330014 u330014   39 Apr 27 16:08 file4_cat
[u330014@localhost Xplore1]$
```

## Removing Directories

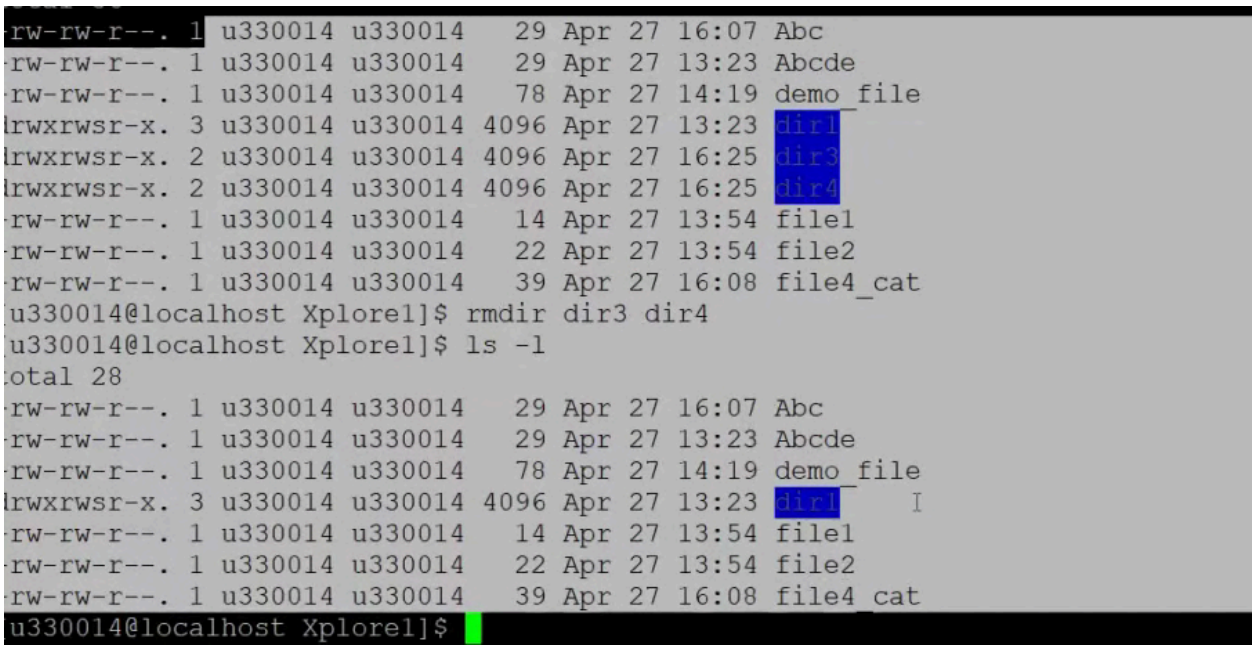
### rmdir command

Directories can be deleted using the rmdir command as follows:

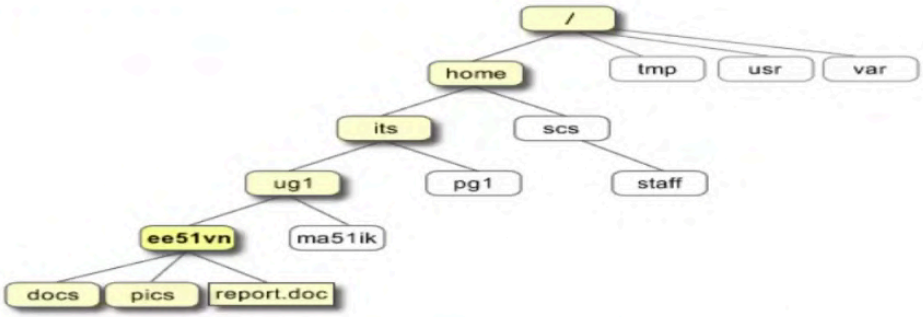
$rmdir dirname

You can remove multiple directories at a time as follows:

$rmdir dirname1 dirname2 dirname3



## Absolute Path and Relative path




**$ cd /home/its/ug1 - Navigate to ug1 directory using absolute path.**
**It starts from the root directory**

**$pwd**
**$/home/its/ug1**
**$cd ee51vn/docs**

**$cd ../../ → will take you to two directories up in hierarchy**

## Searching files using Find

It can search the entire file-system to locate files and directories according to the specific search criteria.

**Syntax : find <search-in directory> <search-parameter> <action>**

**$ find . -name Abc -> searches for file named Abc in current(.) directory and its sub directories**

**$find / -name bin -type d -> searches for directory named bin in root(/) directory and its sub directories**

**$ find . -name "Abc*.??t" -> searches for files starting with Abc and have extension thatends with "t" in the current(.) directory and subdirectories.**

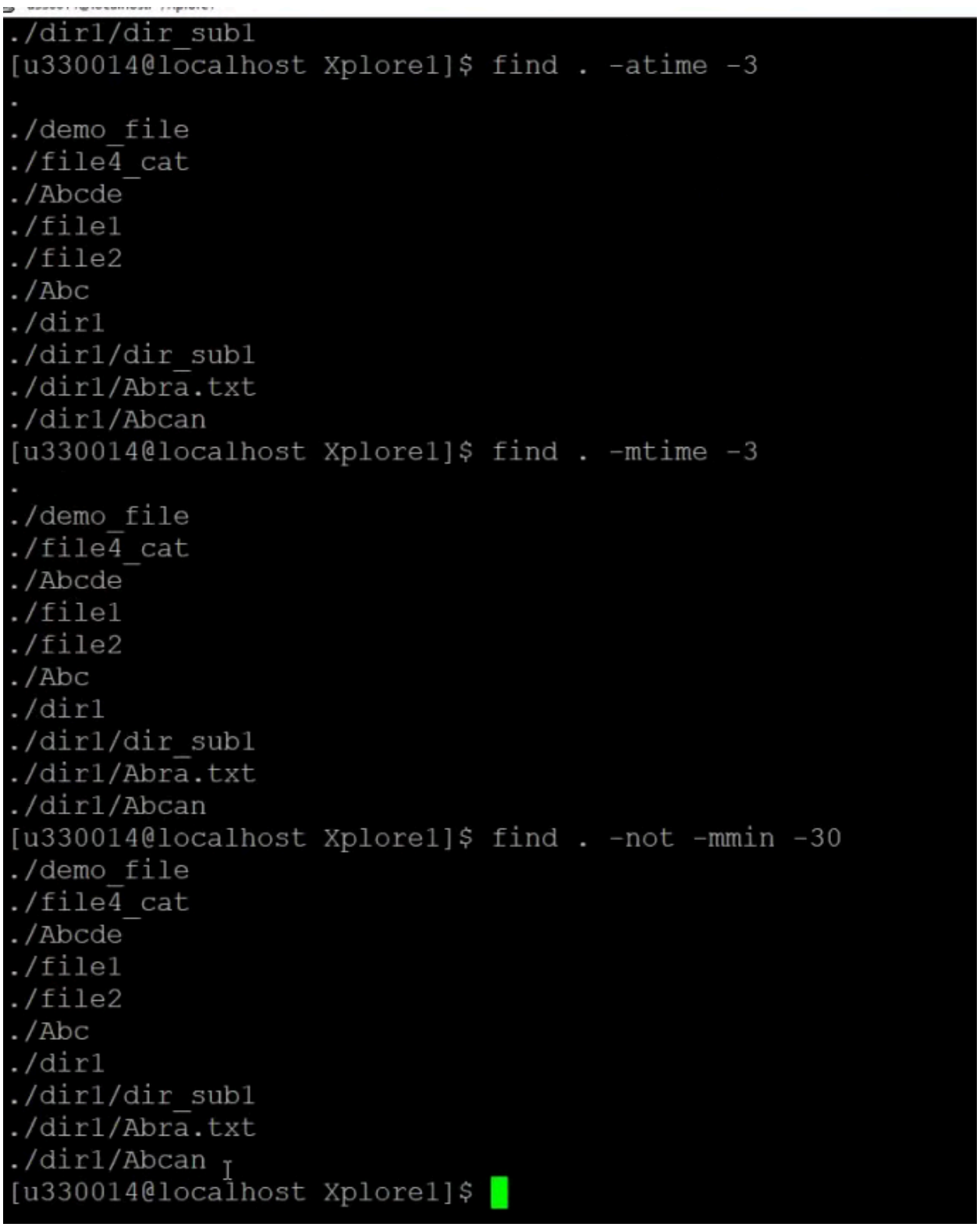
**$find Dir1 Dir2 -name "Abc.*" -> searches for files name Abc with any extension in directories Dir1 and Dir2 only.**

**$ find . -size +10M --> Search for files with size >10MB**

**$ find . -atime -3 --> searches for files which were accessed within last 3 days.**

**$find . -not -mmin -30 --> searches files which are not modified within last 30 minute**

```
[u330014@localhost Xplore1]$ find . -name Abc
./Abc
[u330014@localhost Xplore1]$ find . -name "Abc*"
./Abcde
./Abc
./dir1/Abcan
[u330014@localhost Xplore1]$ find . -name "file?"
./file1
./file2
[u330014@localhost Xplore1]$ find . -name dir* -type d
./dir1
[u330014@localhost Xplore1]$ find . -name *dir* -type d
./dir1
[u330014@localhost Xplore1]$
```




**Kuchh bhi command ya command k baare m anne hetu man command ka use kre**

## Searching files using Locate

It searches for the file in the current DB of files instead of whole file system
So, the search is very fast.

**Syntax : locate <option> <pattern>**

**$ locate -c "Abc" -> gives the count of files with name "Abc"**

**$ locate -e "Abc" -> print only those entries that correspond to existing files.**

**$ locate -i "Abc" -> For case insensitive search**

**$ locate -r "Abc"-> Prints the location of file name "Abc"**

## Diff command to find out difference between two files

Used to display the differences in the files by comparing the files line by line.

**Syntax : diff [options] File1 File2**

**$ diff file1 file2 -> Find difference between 2 files using a and d**

**$ diff -c file1 file2 -> To view differences in context mode**

**$ diff -u file1 file2 -> To view differences in unified mode**

```
[u330014@localhost Xplore1]$ cat Abc
Hello
Difference
How are you
[u330014@localhost Xplore1]$ cat Abcde
Hello
How are you
Difference
[u330014@localhost Xplore1]$ diff Abc Abcde
2d1
< Difference
3a3
> Difference
```

a - add d - delete

```
[u330014@localhost Xplore1]$ diff -c Abc Abcde
*** Abc 2020-04-27 16:07:02.028052026 +0530
--- Abcde       2020-04-27 13:23:06.379053921 +0530
***************
*** 1,3 ****
  Hello
- Difference
  How are you
--- 1,3 ----
  Hello
  How are you
+ Difference
```

```
[u330014@localhost Xplore1]$ diff -u Abc Abcde
--- Abc 2020-04-27 16:07:02.028052026 +0530
+++ Abcde       2020-04-27 13:23:06.379053921 +0530
@@ -1,3 +1,3 @@
 Hello
-Difference
 How are you
+Difference
```

## Cmp command to compare two files

used to compare the two files byte by byte and helps you to find out whether the two files are identical or not.

**Syntax : cmp [OPTION]... FILE1 [FILE2 [SKIP1 [SKIP2]]]**

**$ cmp file1 file2 -> displays the byte and line at which the files are different**

**$ cmp -b file1 file2 -> displays the differing bytes in the output**

**$ cmp -i 10 file1 file2 -> skips a particular number of initial bytes from both the files and then after skipping it compares the files.**

**$ cmp -i 10:12 file1 file2 -> skips a particular number of initial bytes from both the files separately and then it compares the files.**

**$cmp -l file1.txt file2.txt -> prints byte position and byte value for all differing bytes.**

**$ cmp -n 5 file1.txt file2.txt -> to compare first few bytes from both the files.**

```
[u330014@localhost Xplore1]$ cmp Abc Abcde
Abc Abcde differ: byte 7, line 2
[u330014@localhost Xplore1]$ cmp -b Abc Abcde
Abc Abcde differ: byte 7, line 2 is 104 D 110 H
```

```
[u330014@localhost Xplore1]$ cat file1
no intimation
[u330014@localhost Xplore1]$ cat file2
this is the situation
[u330014@localhost Xplore1]$ cmp -b file1 file2
file1 file2 differ: byte 1, line 1 is 156 n 164 t
[u330014@localhost Xplore1]$ cmp -i 11 file1 file2
file1 file2 differ: byte 1, line 1
[u330014@localhost Xplore1]$ cmp -i 8:16 file1 file2
[u330014@localhost Xplore1]$
```

Doesn't return anything if same

## Comm command to compare two files

Compares two sorted files line by line and write to standard output; the lines that are common and the lines that are unique

**Syntax : comm [OPTION]... FILE1 FILE2**

**$comm file1.txt file2.txt -> compares the files and displays common and unique lines**

**$comm -1 file1 file2 -> suppresses unique lines from 1st file**

**$comm -2 file1 file2 -> suppresses unique lines from 2nd file**

**$comm -3 file1 file2 -> suppresses common lines from both the files**

**$comm --nocheck-order abc abcde -> compares non sorted files also**

```
[u330014@localhost Xplore1]$ comm Abc Abcde
                Hello
Difference
                How are you
comm: file 2 is not in sorted order
        Difference
[u330014@localhost Xplore1]$ comm --nocheck-order Abc Abcde
                Hello
Difference
                How are you
         Difference
[u330014@localhost Xplore1]$
```

## Vi editor

It is a visual editor used to enter and edit text files. Invoking vi with/without filename puts it in command mode.

**Syntax :** `vi <filename>`

vi works in three different modes:

- Edit Mode -where any key is entered as text
- Command Mode -where keys are used as commands
- Ex Mode -ex commands can be entered in last line to act on text

Few Special command keys – h,j,k,l (to move left/up/down/right a line)
X, x, dd (deletion of characters and line)

### 1. Modes in `vi` Editor

| Mode | Description |
|---|---|
| **Normal Mode** | Default mode: for navigation, deleting, copying, etc. |
| **Insert Mode** | For typing/editing text |
| **Command Mode** | For saving, quitting, searching, etc. (starts with `:`) |

### 2. Switching Between Modes

| Press Key | Switches To | Description |
|---|---|---|
| `i` | Insert mode | Insert before the cursor |
| `I` | Insert at line start | |
| `a` | Insert after the cursor | |
| `A` | Insert at line end | |
| `Esc` | Normal mode | Return to normal mode from insert |
| `:` | Command mode | Used for save, quit, etc. |

### 3. Basic Insert Commands

| Key | Description |
|---|---|
| `i` | Insert before cursor |
| `a` | Append after cursor |
| `o` | Open new line **below** and insert |
| `O` | Open new line **above** and insert |

### 4. Navigation Commands (Normal Mode)

| Key | Description |
|---|---|
| `h` | Move left |
| `l` | Move right |

| | |
|---|---|
| `j` | Move down |
| `k` | Move up |
| `0` | Start of line |
| `$` | End of line |
| `w` | Next word |
| `b` | Previous word |
| `G` | Go to last line |
| `gg` | Go to first line |
| `:` + `n` | Go to line number `n` |

**5. Editing Commands (Normal Mode)**

| Key | Description |
|---|---|
| `x` | Delete character under cursor |
| `dd` | Delete entire line |
| `dw` | Delete word |
| `yy` | Copy (yank) current line |
| `p` | Paste below cursor |
| `P` | Paste above cursor |
| `u` | Undo |
| `Ctrl + r` | Redo |
| `r` | Replace a single character |
| `R` | Replace text until `Esc` is pressed |

### 6. Search in `vi`

| Command | Description |
|---|---|
| `/word` | Search **forward** for "word" |
| `?word` | Search **backward** for "word" |
| `n` | Go to next match |
| `N` | Go to previous match |

### 7. Save and Quit (Command Mode – after pressing `:`)

| Command | Description |
|---|---|
| `:w` | Save (write) file |
| `:q` | Quit |
| `:wq` or `ZZ` | Save and quit |
| `:q!` | Quit without saving (force quit) |
| `:x` | Save **if** changes made, then quit |

### 8. Miscellaneous

| Command | Description |
|---|---|
| `:set nu` | Show line numbers |
| `:set nonu` | Hide line numbers |
| `.` | Repeat last command |

## Stream editor(sed)

It is a stream editor used to perform basic text transformations on an input stream (a file, or input from a pipeline).

**Syntax : sed -n 'ADDRESS'p filename**

**$ sed -n '3p' demo_file -> prints third line of input file**

**$sed -n '3~2p' demo_file -> prints every 2nd line starting from the line 3**

**$ sed -n '4,8p' demo_file -> prints from 4th line to 8th line from input file**

**$ sed -n '$'p demo_file -> prints only the last line**

**Syntax : sed -n '/pattern/p' filename**

**$sed -n /QL/p demo_file -> prints lines containing “QL”**

**$ sed -n '/PLSQL/,/UI/p' demo_file -> prints all the lines between PLSQL and UI**

```
[u330014@localhost Xplore1]$ cat demo_file
1. SQL
2. PLSQL
3. UNIX
4. UI
5. JAVA
6. Python
7. DataWarehouse
8. .NET - C#
[u330014@localhost Xplore1]$ sed -n '3'p demo_file
3. UNIX
[u330014@localhost Xplore1]$ sed -n '3,6'p demo_file
3. UNIX
4. UI
5. JAVA
6. Python
[u330014@localhost Xplore1]$ sed -n '3~2'p demo_file
3. UNIX
5. JAVA
7. DataWarehouse
[u330014@localhost Xplore1]$ sed -n '$'p demo_file
8. .NET - C#
[u330014@localhost Xplore1]$ sed -n /QL/p demo_file
1. SQL
2. PLSQL
[u330014@localhost Xplore1]$ sed -n '/PLSQL/,/JAVA/p' demo_file
2. PLSQL
3. UNIX
4. UI
5. JAVA
[u330014@localhost Xplore1]$
```

## substitution operation in sed

**Syntax** : $sed 'PATTERNs/REGEXP/REPLACEMENT/FLAGS' filename

- s is substitute command
- / is a delimiter
- REGEXP is regular expression to match
- REPLACEMENT is a value to replace

$ sed 's/UI/UI-HTML,JS/' demo_file

- Prints the content by replacing UI with UI-HTML,JS

## Deletion operation in sed

- In sed the d command is used to delete the pattern space buffer and immediately starts the next cycle.

### syntax

- sed nd filename - 'nd' deletes the nth line and prints the other lines.
- $ sed 'nd' filename

### Example

```
[pts/0][12:04:16:e379230@inchnilp02 ] ~> sed 3d sedfile
1. Linux - Sysadmin, Scripting etc.
2. Databases - Oracle, mySQL etc.
4. Security (Firewall, Network, Online Security etc)
5. Storage
6. Cool gadgets and websites
7. Productivity (Too many technologies to explore, not much time available)
8. Website Design
9. Software Development
10.Windows- Sysadmin, reboot etc.
```

### To delete the last line from input.

- $ sed '$d' filename

## Unix OS – Filters

- Filters

- There are some linux commands that accept input from standard input or files,perform some manipulation on it and produce some output to the standard output.

- Since these commands perform filtering operations on data ,they are called as filters.

## Unix OS – Filters – H

**Head**

- This command display the part of the file
- General format is,
- Head [options] <filename>

Where options can be,

- -n print the first num lines
- -q never print headers identifying file names.
- # By default it gives 10

- *root@ubuntu:/var/log# head -5 syslog*

*Dec 28 05:29:02 ubuntu rsyslogd: [origin software="rsyslogd" swVersion="7.4.4" x-pid="547" x-info="http://www.rsyslog.com"] start*
*Dec 28 05:29:02 ubuntu rsyslogd: rsyslogd's groupid changed to 104*
*Dec 28 05:29:02 ubuntu rsyslogd: rsyslogd's userid changed to 101*
*Dec 28 05:29:02 ubuntu kernel: [ 0.000000] Initializing cgroup subsys*

## Unix OS – Filters –

**Tail**

- The tail command, as the name implies, print the last N number of data of the given input. By default it prints the last 10 lines of the specified files. If more than one file name is provided then data from each file is precedes by its file name.
-

General format is,

- Tail [options] <filename>

Where options can be,

- -n num Prints the last 'num' lines.
- -f shows the last ten lines of a file and will update when new lines are added.

**Example Commands**

- **Tail -10 syslog**
- **Tail +10 syslog**
- **Tail -10f syslog**

## Unix OS – Filters – Tee

**Tee**

The tee command reads from the standard input and writes to both standard output and one or more files at the same time. tee is mostly used in combination with other commands through piping.

General format is,

```
Tee [OPTIONS] [FILE]
```

Where options can be,

-a (--append) - Do not overwrite the files instead append to the given files.

-i (--ignore-interrupts) - Ignore interrupt signals.

**Example Commands**

**Tail -10 syslog | Tee –a syslog_extract**

## Unix OS – Filters – Find

- Find it searches for files and directories in a directory hierarchy based on a user given expression and can perform user-specified action on each matched file.

**Format**
Find [options] [path] [expression],

**Find all files in the entire system that are named test.txt:**
find / -name "test.txt"
**Find all files in the /etc directory not named test.txt:**
find /etc/ -not -name "test.txt"
**Find all directories called log:**
find / -type d -name "log"
**Find all files in the /usr/bin directory or subdirectories that are larger than 1 megabyte:**
find /usr/bin/ -size 1M
**Find all files owned by the user named "chad":**
find / -user chad
**Find all files in the /etc directory owned by the user root, and paginate:**
find /etc/ -user root | more

## Unix OS – Filters – GREP

Ggrep (global search for regular expression)command is used to search specified pattern from a specified file and displays those lines containing the pattern.

**Format**
Grep [options] pattern <filename> Where options can be,

b ignores spaces ,tabs
I ignores distinction matching

V displays only the lines that do not specified pattern.

C displays the total number of occurrence of the pattern in the file.

A2 displays the resultant lines along with their line numbers.

# Unix OS – Filters – GREP – HandsOn

**Example**

**1. grep -A2 “configuration” h*.xml**

httpfs-site.xml:<configuration>
httpfs-site.xml-
httpfs-site.xml:</configuration>

**2. grep -n "<configuration>" h*.xml**

hadoop-policy.xml:**25**:<configuration>
hdfs-site.xml:**19**:<configuration>
httpfs-site.xml:**15**:<configuration>

3. grep -vn "<configuration>" h*.xml

4. grep -c "<configuration>" h*.xml

5. grep -l "<configuration>" h*.xml

6. grep -X "<configuration>" h*.xml

```
viji@test:~/unix_webinar$ cd grep/
viji@test:~/unix_webinar/grep$ grep -n "<configuration>" h*.xml
hadoop-policy.xml:25:<configuration>
hdfs-site.xml:20:<configuration>
httpfs-site.xml:15:<configuration>
viji@test:~/unix_webinar/grep$ ls h*.xml
```

# Unix OS – Filters – Regex Expression

**Regex Expression**

Match zero or more characters. Matches a single character.

- range represented by the characters.

**Bracket Expressions**

- [abcd] matches a single character which is a,b,c or d.
- [A-Z] matches a single character within the range of character from A to Z

**#^ caret symbol to add exception rule**

# [^abcd] matches a single character which is not a,b,c or d.

**Anchor Expression**

- ^<character>matches the lines that are beginning with character specified in <character>.

## Unix OS – Filters – Regex Expression

**Escaping Meta-Characters**

- escape characters by using the backslash character (\) before the character

**Example**

grep "^[A-Z].*\.$" GPL-3

Here . is escape character using \

**Repeat Pattern Zero or More Times**

- [a-d]* matches a string where characters can be inside the a to d range.

**Quantifiers**

- . (dot) - a single character.
- **?** - the preceding character matches 0 or 1 times only.
- * - the preceding character matches 0 or more times.
- **+** - the preceding character matches 1 or more times.

Example

## Unix OS – Filters – Regex Expression

**Grouping**

- When a group of characters is enclosed in parentheses, the next operator applies to the whole group.

**Example**

Grep '^\([Ll]nux|unix\)' filename

## Unix OS – Filters – sed

The sed stream editor is a text editor that performs editing operations on information coming from standard input or a file. Sed edits line-by-line and in a non-interactive way.

**Format**
sed [options] commands [file-to-edit]
**Commands**
p Prints the line
d Deletes the line
s/pattern1/pattern2/ Substitutes the first occurrence of pattern1 with pattern2

Options
-n no printing
-e script
-f file

## Unix OS – Filters – sed

Example

1.
```
sed 's/host\-url/123.45.64./' hdfs-site.xml > new-hdfs-site.xml
```

2.
```
sed n 's/host\-url/123.45.64./' hdfs-site.xml > new-hdfs-site.xml
```
3.
```
sed p 's/host\-url/123.45.64./' hdfs-site.xml > new-hdfs-site.xml
```

4
```
sed p 's/host\-url/123.45.64./' hdfs-site.xml > new-hdfs-site.xml
```

Regex
```
sed 's/\(^<m[a-zA-Z].*\)config2/<AWS/' hdfs-site.xml > new2-hdfs-stie.xml
```

# Unix OS – Filters – Sort

- Sort
- This command sorts the contents of a given file based on ASCII values of characters.

- General format is,
- Sort [options] <filename>

Where options can be,

-n, --numeric-sort
compare according to string numerical value

-r, --reverse
reverse the result of comparisons

-u, --unique
sort the file by removing duplicates

-c, --check
check whether input is sorted; do not sort

-k, --key=POS1[,POS2]
start a key at POS1, end it at POS2 (origin 1)

-m, --merge
merge already sorted files; do not sort

# Unix OS – Filters – Sort

**$ cat > employee.txt**

```
manager  5000
clerk    4000
employee  6000
peon     4500
director 9000
guard     3000
```

**$ sort -k 2n employee.txt**

```
guard    3000
clerk    4000
peon     4500
manager  5000
employee 6000
director 9000
```

# Unix OS – Pipe

- The Pipe is a command in Linux that lets you use two or more commands such that output of one command serves as input to the next. In short, the output of each process directly as input to the next one like a pipeline. The symbol '|' denotes a pipe.

- **Example**

1. **Use sort and uniq command to sort a file and print unique values.**

   *sort record.txt | uniq*

2. **Use head and tail to print lines in a particular range in a file**

$ cat sample2.txt | head -7 | tail -5

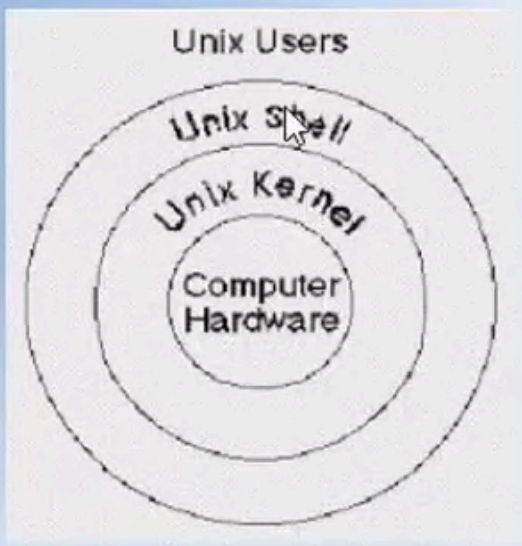
3. **Use cat, grep, tee and wc command to read the particular entry from user and store in a file and print line count.**

$ cat weblog.txt | grep "Rajat Dua" | tee file2.txt | wc -l

## Components of UNIX OS

**The main components that unites all versions of UNIX is the following four basics:**

- Kernel
- Shell
- Commands and Utilities
- Files and Directories




## kernel

The kernel is the heart of the operating system. It interacts with hardware and most of the tasks like memory management, time scheduling and file management.

**WHAT KERNEL DOES?**

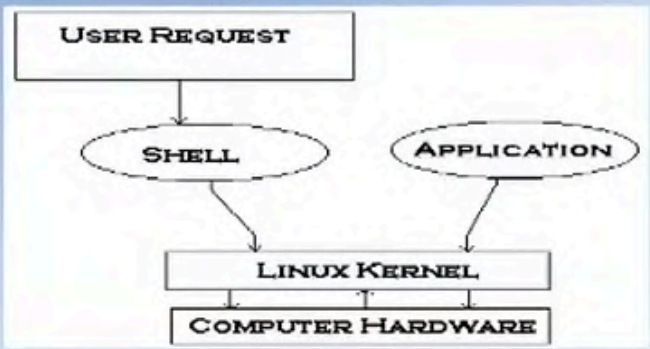
- Directly communicates with hardware
- Makes Unix portable
- Allocates the resources to different users and tasks
- Keeps track of various programs being run, allocating time to each, depending when to stop one and start another.
  - Assigns spaces for files
  - Runs the shell programs

kernel runs the shell program

# shell

**Shell:**

Users communicate with the kernel through a program known as the shell. The shell is a command line interpreter, it translates commands entered by the user and converts them into a language that is understood by the kernel.




**Commands and Utilities:**

- There are various command and utilities which you would use in your day to day activities. **cp** , **mv**, **cat** and **grep** etc. are few examples of commands and utilities.
- There are over 250 standard commands available in Unix. Some of the commands are explained below in detail.

**Files and Directories:**

- All data in UNIX is organized into files.
- All files are organized into directories.
- These directories are organized into a tree-like structure called the file system.

# ADVANTAGES

**1.Multi tasking:**

A user can also run multiple programs at the same time, hence UNIX is called multitasking.

**2.Multiuser Capabilities:**

•Several people can use a UNIX computer at the same time, hence UNIX is called a multiuser system.

•Saves time by allowing more than one person to work on a set of information at a time.

•More than one terminal can be connected to one computer and the users of the terminals can all run programs, access files and print documents.

**3.Portability:**

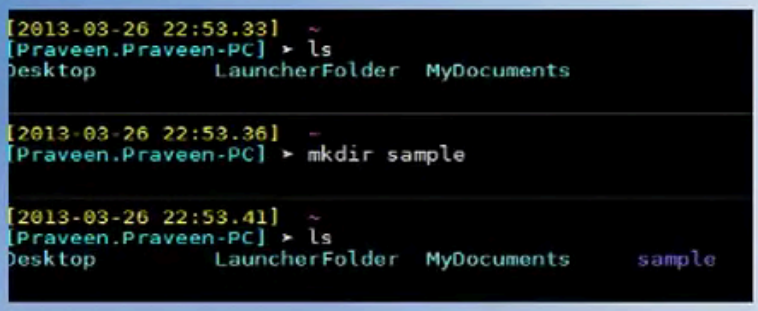
The System is written in a high level language making it easier to read, understand, change and move to other machines.

# Listing files and directories

**Listing files and directories**

- When you first login, your current working directory is your home directory.
- Your home directory has the same name as your user-name, for example, **ee735618ab**, and it is where your personal files and subdirectories are saved.
- To find out what is in your home directory, type

  **% ls**
- The **ls** command ( lowercase L and lowercase S ) lists the contents of your current working directory.

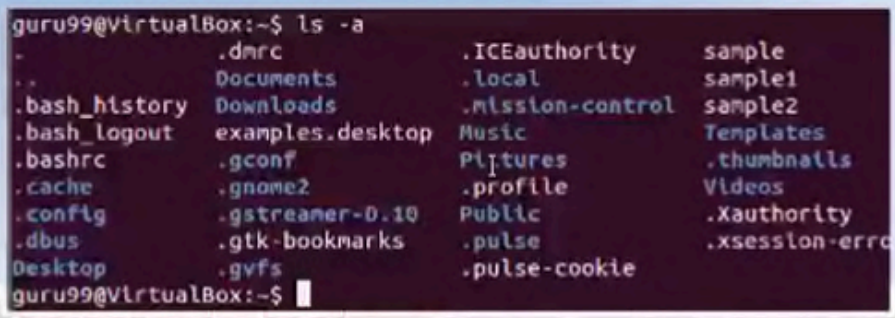



- There may be no files visible in your home directory, in which case, the UNIX prompt will be returned. Alternatively, there may already be some files inserted by the System Administrator when your account was created.
- **ls** does not, in fact, cause all the files in your home directory to be listed, but only those ones whose name does not begin with a dot (.)

- Files beginning with a dot (.) are known as **hidden files** and usually contain important program configuration information. They are hidden because you should not change them unless you are very familiar with UNIX.
- To list all files in your home directory including those whose names begin with a dot type.

  % ls -a

  As you can see, **ls -a** lists files that are normally hidden.



- **ls** is an example of a command which can take options: **-a** is an example of an option. The options change the behavior of the command.

  ls –l:

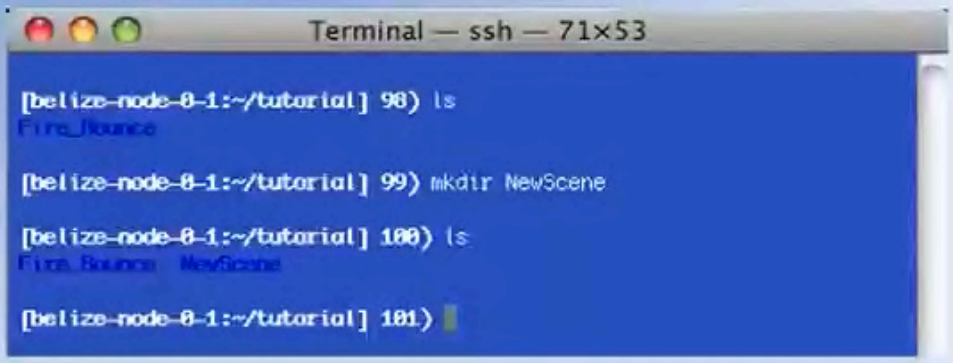
- The command **ls** supports the **-l** option which would help you to get more information about the listed files

## Making Directories

**mkdir (make directory)**

We will now make a subdirectory in your home directory to hold the files you will be creating. To make a subdirectory called **newscene** in your current working directory type

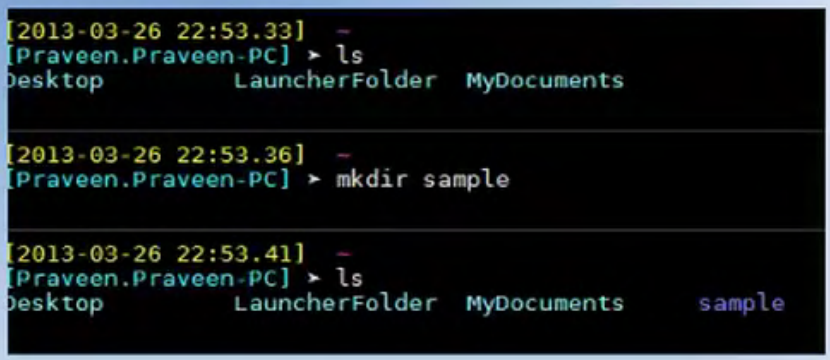
% mkdir newscene



% mkdir sample

To see the directory you have just created, type

% ls




## Changing to a different directory

**cd (change directory)**

- The command **cd** ***directory*** means change the current working directory to *'directory'*.
- The current working directory may be thought of as the directory you are in, i.e. your current position in the file-system tree.
- To change to the directory you have just made, type

  % cd sample

- Type **ls** to see the contents.

## The directories . and ..

- Still in the **sample** directory, type

  % ls -a

- As you can see, in the **sample** directory (and in all other directories), there are two special directories called (.) and (..)

**The current directory (.)**

- In UNIX, (.) means the current directory, so typing

  % cd .

  NOTE: there is a space between cd and the dot

- (cd .) means stay where you are (the **sample** directory).

**The parent directory (..)**

•(..) means the parent of the current directory, so typing

% cd ..

•will take you one directory up the hierarchy (back to your home directory).

•Note: typing **cd** with no argument always returns you to your home directory.

## Pathnames

**pwd (print working directory)**

•Pathnames enable you to work out where you are in relation to the whole file-system.

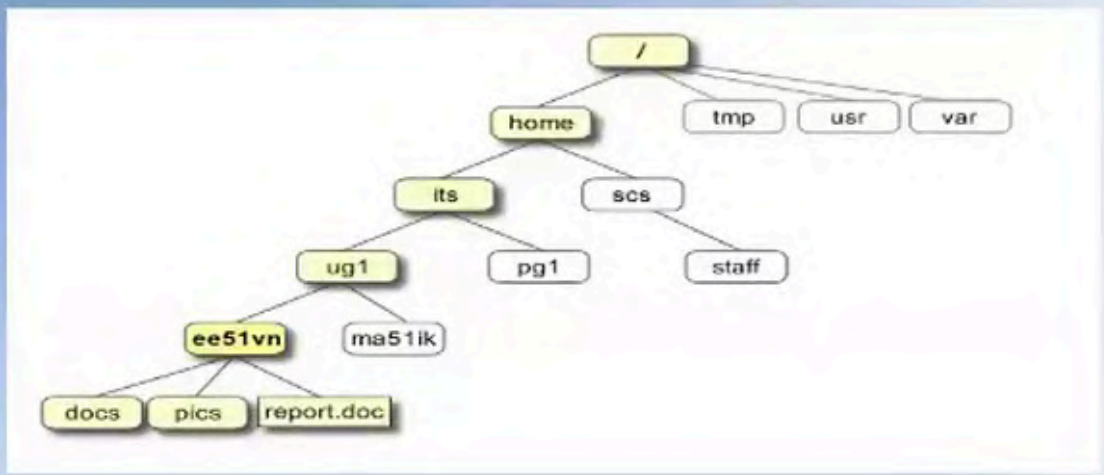
•For example, to find out the absolute pathname of your home-directory, type **cd** to get back to your home-directory and then type

% pwd

•The full pathname will look something like this –

**/home/its/ug1/ee51vn**

which means that **ee51vn** (your home directory) is in the sub-directory **ug1** (the group directory),which in turn is located in the **its** sub-directory, which is in the **home** sub-directory, which is in the top-level root directory called " / " .



## Copying Files

**cp (copy)**

When copying files from one directory to another you need to know for certain which directory is the current working directory.

If you are not sure ,use the pwd command

***Make a copy in the current directory***

cp oldfilename newfilename

Eg: cp file1.html file2.html

## Moving files

***Move a file into a sub-directory***

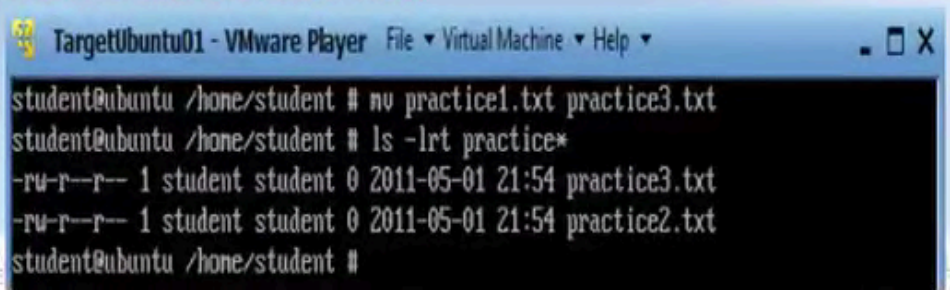
mv filename dir-name

- The cp command makes a copy of the file with the new name or in the new location, and leaves the original file in place.
- If you use the mv command the file is moved, so the original is deleted and the only copy is the one in the new location.

***Re-naming a file***

- The mv command can also be used to rename a file while leaving it in the same directory.

mv oldfilename newfilename



## Removing files and directories

**rm (remove), rmdir (remove directory)**

To delete (remove) a file, use the **rm** command.

As an example, we are going to create a copy of the **science.txt** file then delete it.

Inside your **sample** directory, type

```
% cp science.txt tempfile.txt
% ls
% rm tempfile.txt
% ls
```

You can use the **rmdir** command to remove a directory (make sure it is empty first).

## Displaying the contents of a file on the screen

**clear (clear screen)**

•Before you start the next section, you may like to clear the terminal window of the previous commands so the output of the following commands can be clearly understood.

```
% clear
```

This will clear all text and leave you with the % prompt at the top of the window.

**cat (concatenate)**

•The command cat can be used to display the contents of a file on the screen.

```
% cat science.txt
```

As you can see, the file is longer than the size of the window, so it scrolls past making it readable.

**head**

- The **head** command used to display the beginning of a text file or piped data to the screen.
- First clear the screen then type

```
% head science.txt
```

**tail**

- The **tail** command used to display the bottom of a text file or piped data to the screen.
- Clear the screen and type

```
% tail science.txt
```

## Searching the contents of a file

**grep**

•**GREP (Global search of REgular expressions and Print)** is one of many standard UNIX utilities. It searches files for specified words or patterns.

```
% grep science science.txt
```

•As you can see, **grep** has printed out each line containing the word **science**.

•To ignore upper/lower case distinctions, use the -i option, i.e. type

```
% grep -i science science.txt
```

To search for a phrase or pattern, For example to search for spinning top, type quotes (the apostrophe symbol).

```
% grep -i 'spinning top' science.txt
```

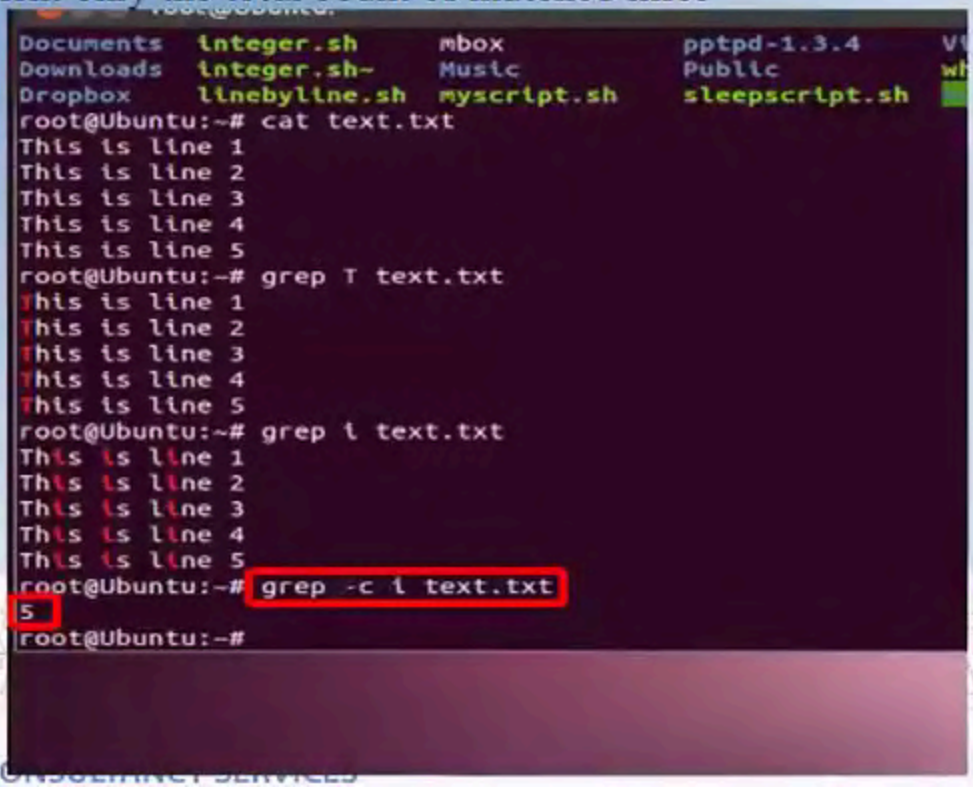
# grep

Some of the other options of grep are:

-v display those lines that do NOT match pattern

-n proceed each matching line with the line number

-c print only the total count of matched lines



## Searching the contents of a file

**wc (word count)**

The **wc** (word count) command in Unix operating systems is used to find out number of **newline count**, **word count**, **byte and characters** count in a files specified by the file arguments. The syntax of **wc** command as shown below.

**% wc [options] filename**

- To display the number of words in a file

  **% wc -w science.txt**

- To find out how many lines the file has, type

  **% wc -l science.txt**

- To display the count of characters from a file

  **%wc –m science.txt**

- To display the length of the longest line in a file

  **%wc –L science.txt**

## Getting Help

**On-line Manuals**

- There are on-line manuals which gives information about most commands. The manual pages tell you which options a particular command can take, and how each option modifies the behaviour of the command.

- Type **man** command is to read the manual page for a particular command.

- For example, to find out more about the **wc** (word count) command, type

  **% man wc**

Alternatively

**% whatis wc**

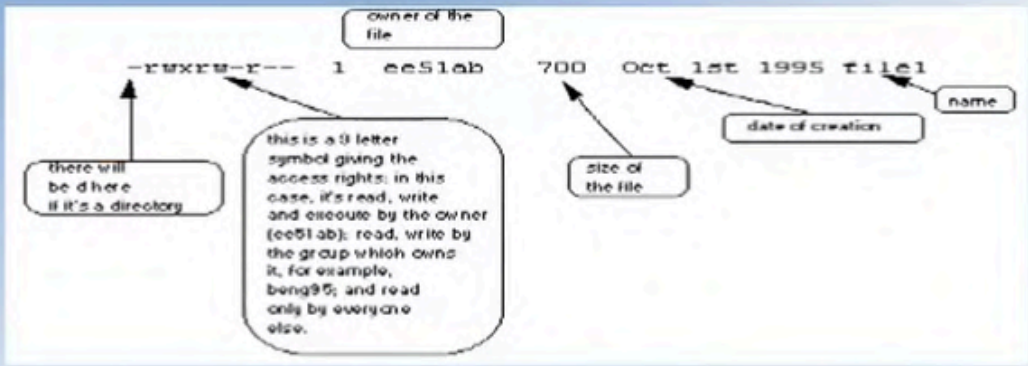
gives a one-line description of the command.

## File system security (access rights)

In your **unixstuff** directory, type

% ls -l **(l for long listing!)**

•You will see that you now get lots of details about the contents of your directory, similar to the example below.



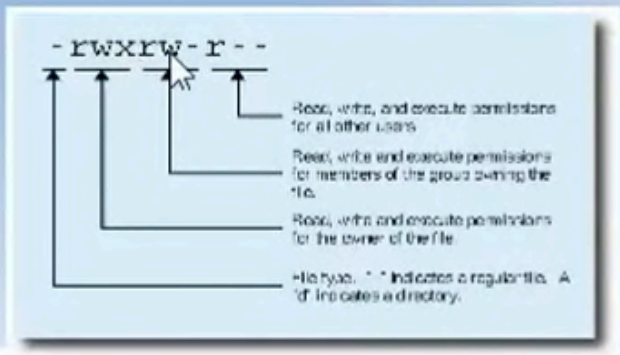

## File system security (access rights)

- Each file (and directory) has associated access rights, which may be found by typing **ls -l**. Also, **ls -lg** gives additional information as to which group owns the file (beng95 in the following example):

```
-rwxrw-r-- 1 ee51ab beng95 2450 Sept29 11:52 file1
```

- In the left-hand column is a 10 symbol string consisting of the symbols d, r, w, x, -. If d is present, it will be at the left hand end of the string, and indicates a directory: otherwise - will be the starting symbol of the string.

- The 9 remaining symbols indicate the permissions, or access rights, and are taken as three groups of 3.
- The **left** group of 3 gives the file permissions for the user that owns the file (or directory) (ee51ab in the above example);
- The **middle** group gives the permissions for the group of people to whom the file (or directory) belongs (eebeng95 in the above example);
- The **rightmost** group gives the permissions for all others.
- The symbols r, w, etc., have slightly different meanings depending on whether they refer to a simple file or to a directory.




## Changing access rights

**chmod (changing a file mode)**

•Only the owner of a file can use chmod to change the permissions of a file. The options of chmod are as follows

| Symbol | Meaning |
|---|---|
| u | User |
| g | Group |
| o | Other |
| a | All |
| r | Read |
| w | write (and delete) |
| x | execute (and access directory) |
| + | add permission |
| - | take awaypermission |

# Vi editor

There are many ways to edit files in Unix and one of the best ways is using screen-oriented text editor **vi**. This editor enable you to edit lines in context with other lines in the file.
Now a days you would find an improved version of vi editor which is called **VIM**. Here VIM stands for **ViIMproved.**

| | |
|---|---|
| **vi filename** | Creates a new file if it already does not exist, otherwise opens existing file. |
| **vi -R filename** | Opens an existing file in read only mode. |
| **view filename** | Opens an existing file in read only mode. |

# Some of the useful characters used in vi editor

| Command | Description |
|---|---|
| i | Inserts text before current cursor location. |
| I | Inserts text at beginning of current line. |
| a | Inserts text after current cursor location. |
| A | Inserts text at end of current line. |
| o | Creates a new line for text entry below cursor location. |
| O | Creates a new line for text entry above cursor location. |
| x | Deletes the character under the cursor location. |
| X | Deletes the character before the cursor location. |
| dd | Delete a line |
| 3dd | Delete 3 lines |
| ←↑↓→ | arrow keys move the cursor |
| hjkl | Same as arrow keys |

# The Unix Shell

- The shell is your interface with the Unix system.

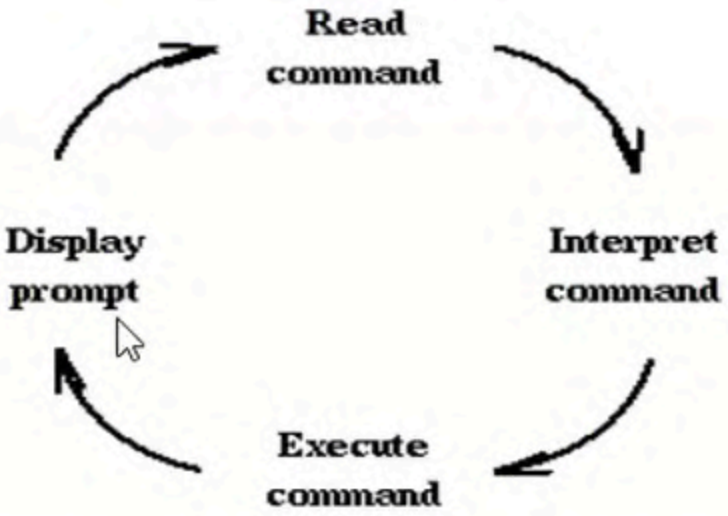
- Is an "interpreter" of commands
- When a group of UNIX command to be executed regularly ,then they are stored in a file, such files are called Shell Script, Shell Programs or Shell Procedure
- no restriction on the extension but it is a common practice to use ".sh" extension for shell script program.

- Commands
  - ✓ Built in subroutines (do not require the kernel to start another process to run them)
  - ✓ Non built in commands (require the kernel to create a new child process to perform the command). The shell waits until the child process finishes before accepting the next command.

```
1.
echo Hello World
**********************
2.
echo "Printing text with newline"
echo -n "Printing text without newline"
echo -e "\nRemoving \t backslash \t characters\n"
**********************
3.
# Add two numeric value
((sum=25+35))

#Print the result
echo $sum
**********************
```

# The Unix Shell




Interpreter
loop

# Shells – Entering Commands

1. Introducing Shells – entering commands, history, wild cards
2. Introducing Shell scripts – alternatives, tools, input/output, variables, comments, continuation of lines, magic line
3. Shell script constructs – using variables, for, while, if, test, case
4. Interaction with the Environment – environment variables
5. Best practices – comments, error messages, debugging
6. Why write scripts? – complicated commands, data management, automation

# Shell Script – What & Why?

1. What is Shell Script?

   Shell Script is series of command written in plain text file. Shell script is just like batch file in MS-DOS but have more power than the MS-DOS batch file

2. Why Shell Script?

- Automate frequently run tasks
- Remember complicated command/command-line options
- Filter through data and logs

# Shell Script – 3 simple steps

1. Use any editor like vi to write shell script.
2. After writing shell script set execute permission for your script.

   Syntax: chmod <permission> <your-script-name>

   Example:

   $chmod +x your-script-name

   $chmod 755 your-script-name
3. Execute your script

   Syntax:

   bash <your-script-name>

   sh <your-script-name>

   ./<your-script-name>

# Shell Script – Wild cards

1. * - Matches any string or group of characters.
   - ls ut*.c – will show all files having extension .c but file name must begin with "ut".
2. ? - Matches any single character.
   - ls fo? - will show all files whose names are 3 character long and file name begin with "fo"
3. […] - Matches any one of the enclosed characters
   - ls [abc]* - will show all files beginning with letters a,b,c
4. [..-..] – Matches range of character

   ls [a-d]* - will show all files beginning with letters a,b,c,d

   ls [^a-c]* - will show all files beginning with letters other than a,b,c

# Shell Script – Echo Command

1. Use echo command to display text or value of variable.
2. echo [options] [string, variables...]

   Options

   -n Do not output the trailing new line.

   -e Enable interpretation of the following backslash escaped characters in the strings:

   \b backspace

   \c suppress trailing new line

   \n new line

   \t horizontal tab

   \\ backslash

Examples

1. Output 'Hello World'.
   - echo "Hello World"
2. Output 'Hello World' with no new line
   - echo –n "Hello World"

# Shell Script – User Defined Variables

1. To define user defined variable (UDV) use following syntax

   Syntax: variable_name=value

   Example:

   To define variable called 'vehicle' having value Bus

   vehicle1=Bus

   vehicle2="A Bus"

   echo "The value of vehicle1 and 2 is :$vehicle1:$vehicle2:"

   To define variable called n having value 10

   n=10

   echo "The value of n is $n"

   Note: Syntax for accessing/printing UDV is $variable_name

# Shell Script – Evaluation of UDV

| | |
|---|---|
| $var | value of var; nothing if var undefined |
| ${var} | same; useful if alphanumerics follow variable name |
| ${var-str} | value of var if defined; else str; $var unchanged. |
| ${var=str} | value of var if defined; else str; if undefined, $var set to str. |
| ${var?str} | if defined, $var; else, print str and exit shell; if str empty, print: var:parameter not set |
| ${var+str} | str if $var defined, else null. |

# Shell Script – User Defined Variables

1. Rules for naming variable name
   - Variable name must begin with Alphanumeric character or underscore character (_), followed by one or more Alphanumeric character ( eg. HOME, SYSTEM_VERSION )
   - Don't put spaces on either side of the equal sign when assigning value to variable. ( number=10 )
   - Variables are case-sensitive, just like filename in Linux. (no, No, NO, nO are all different variable names)
   - You can define NULL variable as follows
     - vech= (or) vech=""
   - Do not use ?, * etc. for naming variables.

# Shell Script – Referencing Variables

1. Variable is referenced by placing dollar sign in front of variable name
   - Syntax: $variablename
2. Variable can be referenced using curly braces.
   - Syntax: ${variablename}

```
#set initial value
myvar=unix
Echo $myvar          #unix
Echo ${myvar}   #unix
Echo {$myvar}   #{unix}
Echo ${myvar}class  #unixclass
```

# Shell Script – More…

1. Exit Status of command or shell script?

   echo $?  ( zero return value indicates success, else failure )
2. Read statement
   - Use to get input (data from user) from keyboard and store (data) to variable.

     Syntax:

     read variable1

Example

```
echo –n "Please enter your name: "
read NAME
echo "Welcome! $NAME !"
```

2. To run two command with one command line

   Syntax:

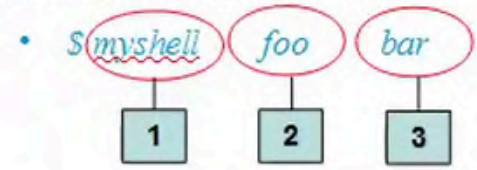
   command1;command2

Example

```
date;who
```

# Shell Script – Command Line Arg.

1. Tells the command/utility which option to use.
2. Informs the utility/command which file or group of files to process (reading/writing of files).




1 Shell Script name i.e. myshell ($0)

2 First command line argument i.e. foo ($1)

3 Second command line argument i.e. bar ($2)

$# - No. of Arguments (Here, 2) $* - Expands all argument



# Input/Output Redirection

- There are three types of file descriptors
  - ✓ Standard Input (0)
  - ✓ Standard Output (1)
  - ✓ Standard Error (2)
- Input/Output redirection connects processes with files.

# Shell Script – Input/Output redirection

1. > - To output result (output of command or shell script) to file
   - Note: If file already exists, it will be overwritten else new file is created

2. >> - To output result to END of file
   - Note: If file exists , it will be opened and new information/data will be written to END of file, without losing previous information/data
   - If file does not exist, then new file is created

3. < - To take input from file instead of key-board

5.

```
valid=true
count=1
while [ $valid ]
do
echo $count
if [ $count -eq 5 ];
then
break
fi
((count++))
done
```

**********

6.

```
for (( counter=10; counter>0; counter-- ))
do
echo -n "$counter "
done
printf "\n"
```

fi is end of if

## Structured Language Construct – For Loop

Syntax:

```
for { variable name } in { list }
do
        command1
        command2
        …
        last command
done
```

Example

```
#print welcome 5 times
for i in 1 2 3 4 5
do
    echo "Welcome $i times"
done
```

## Structured Language Construct – For Loop

Looping over files

```
# Example to look at files in a directory and print out file names
for filename in *
do
    echo "$filename"
done
# print out the file names ending in .doc
for filename in *.doc
do
    echo "$filename"
done
```

# Structured Language Construct – For Loop

Nested for loops for fixed number of iterations

```
# Nested for loop
for i in 1 2 3 4 5
do
    echo –n “Row $i: ”
    for j in 1 2 3 4 5
    do
            sleep 1             # sleep for a second
            echo –n “$j “
    done
    echo                        # output new line
done
```

```
**********
6.
for (( counter=10; counter>0; counter-- ))
do
echo -n "$counter "
done
printf "\n"
**********
7.

echo "Enter Your Name"
read name
echo "Welcome $name to Shell"
**********
```

8.

```
n=1
if [ $n -lt 10 ];
then
echo "It is a one digit number"
else
echo "It is a two digit number"
fi
```

**********

9.

```
echo "Enter username"
read username
echo "Enter password"
read password

if [[ ( $username == "admin" && $password == "secret" ) ]]; then
echo "valid user"
else
echo "invalid user"
fi
```

**********

main.sh saved

```
1   echo "Enter any number"
2   read n
3
4   if [[ ( $n -eq 15 || $n  -eq 45 ) ]]
5   then
6   echo "You won the game"
7   else
8   echo "You lost the game"
9   fi
```

https://TalkativeFlimsyEven

```
GNU bash, version 4.4
bash main.sh
Enter any number
15
You won the game
```

main.sh saved

```
1   echo "Enter your lucky number"
2   read n
3
4   if [ $n -eq 101 ];
5   then
6   echo "You got 1st prize"
7   elif [ $n -eq 510 ];
8   then
9   echo "You got 2nd prize"
10  elif [ $n -eq 999 ];
11  then
12  echo "You got 3rd prize"
13
14  else
15  echo "Sorry, try for the next time"
16  fi
```

https://TalkativeFlimsyEvents

```
GNU bash, version 4.4.12(1)-rel
bash main.sh
Enter your lucky number
101
You got 1st prize
```

```
13.
string1="Unix"
string2="Hint"
echo "$string1$string2"
string3=$string1+$string2
string3+=" is a good tutorial blog site"
echo $string3
******************************
14.
Str="Learn Linux from LinuxHint"
subStr=${Str:6:5}
echo $subStr
******************************
```


```
**********************************
16.
create file:-
function F1()
{
echo 'I like bash programming'
}

F1

execute:-
bash function_example.sh
**********************************
```


uppr waala function input.sh m banaya hai aur execute main.sh m kiya hai

```
17.Create function with Parameters:
Rectangle_Area() {
area=$(($1 * $2))
echo "Area is : $area"
}

Rectangle_Area 10 20

execute:-
bash function_parameter.sh

**************************************
**************************************
18.Read a file

#!/bin/bash
file='book.txt'
while read line; do
echo $line
done < $file

 bash read_file.sh
cat book.txt
**************************************
```

main.sh saved

```
1  grep TCS text.txt|wc
2
3
4
```

https://TalkativeFlimsyEvents

```
GNU bash, version 4.4.12(1)-release
> bash main.sh
      1       6      28
>
```

## 195. Tenth Line

Solved

Easy | Topics | Companies

Given a text file `file.txt`, print just the 10th line of the file.

Bash | Auto

```
1  # Read from the file file.txt a
2  sed -n '10p' file.txt
```

# 193. Valid Phone Numbers

Solved

Easy | Topics | Companies

Given a text file `file.txt` that contains a list of phone numbers (one per line), write a one-liner bash script to print all valid phone numbers.

You may assume that a valid phone number must appear in one of the following two formats: (xxx) xxx-xxxx or xxx-xxx-xxxx. (x means a digit)

You may also assume each line in the text file must not contain leading or trailing white spaces.

**Example:**

Assume that `file.txt` has the following content:

```
987-123-4567
123 456 7890
(123) 456-7890
```

Your script should output the following valid phone numbers:

```
987-123-4567
(123) 456-7890
```

```
grep -e '\(^[0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}$\)' \
    -e '\(^([0-9]\{3\})[ ]\{1\}[0-9]\{3\}-\([0-9]\{4\}\)$\)' file.txt
```

`grep` option to specify a **pattern**. You use `-e` twice to give two patterns: one for each phone number format.

## Pattern 1 (for `xxx-xxx-xxxx`)

\(^[0-9]\{3\}-[0-9]\{3\}-[0-9]\{4\}$\)

### Let's break it:

| Part | Meaning |
|---|---|
| `\(` and `\)` | Grouping in **basic regex** (escaped parentheses). Optional here. |
| `^` | Start of the line |

| | |
|---|---|
| `[0-9]` | Match any **digit** (0–9) |
| `\{3\}` | Exactly **3 times** (i.e., 3 digits) |
| `-` | A literal hyphen `-` |
| `[0-9]\{3\}` | Match **3 digits** again |
| `-` | Another hyphen |
| `[0-9]\{4\}` | Match **4 digits** |
| `$` | End of the line |
| `\(...\)` | Optional capturing group (used here, but not necessary for `grep`) |

✅ **Matches lines like**: `123-456-7890`

## Pattern 2 (for `(xxx) xxx-xxxx`)

\(^([0-9]\{3\})[ ]\{1\}[0-9]\{3\}-\([0-9]\{4\}\)$\)

### Let's break this too:

| Part | Meaning |
|---|---|
| `^` | Start of line |
| `(` | Literal `(` — no need to escape inside `'...'` here, but okay |
| `[0-9]\{3\}` | Exactly 3 digits |
| `)` | Literal `)` |
| `[ ]\{1\}` | Exactly **1 space**. Could also use `\s` in extended regex. |
| `[0-9]\{3\}` | 3 digits |
| `-` | Hyphen |

| | |
|---|---|
| `\([0-9]\{4\}\)` | Group of 4 digits (captured, but again capturing is not used here) |
| `$` | End of line |

✅ **Matches lines like**: `(123) 456-7890`

## Summary Table of Common Regex Elements:

| Symbol | Meaning in this context |
|---|---|
| `^` | Start of line |
| `$` | End of line |
| `[0-9]` | Any digit from 0 to 9 |
| `\{n\}` | Exactly n times (e.g., `\{3\}` = 3 times) |
| `-` | Literal dash `-` |
| `[ ]\{1\}` | Exactly one space |
| `\( \)` | Grouping in **basic regex** (needed to be escaped) |
| `-e` | Specify a new regex pattern in `grep` |

```
# grep -E '^([0-9]{3}-[0-9]{3}-[0-9]{4}|\([0-9]{3}\)[ ]{1}[0-9]{3}-[0-9]{4})$'  file.txt


grep -E -e '^[0-9]{3}-[0-9]{3}-[0-9]{4}$' \
   -e '^\([0-9]{3}\) [0-9]{3}-[0-9]{4}$' file.txt

```

```
grep -E -e 'pattern1' \
        -e 'pattern2' file.txt
```

## Why the `\` is used here:

The backslash `\` at the **end of the line** means:

> "Continue this command on the next line."

It is **not part of the command or regex**. It's a shell feature.