OST LAB (CSE-2164)

Compiled by,
Prakash K. Aithal

Course Objectives

- Illustrate and explore the basic commands, shell scripting and system calls related to Linux operating system.
- To learn about Open Source programming and debugging tools
- To Learn about Open Source Documentation Tools

Course Outcomes

- Ability to execute Linux commands, shell scripting using appropriate Linux system calls.
- Ability to use development tools and debugging tools effectively
- Ability to develop simple web applications using open Source Technologies

Evaluation Plan

- Internal Assessment Marks : 60
- (5 M for each lab * 8 labs= 40M + Midterm evaluation 20M)
- ENDEXAM Marks:40

Pre-Lab Session Instructions

- Students should carry the Class notes, Lab Manual and the required stationery to every lab session.
- Be in time and follow the Instructions from Lab Instructors.
- Must Sign in the log register provided.
- Make sure to occupy the allotted seat and answer attendance.
- Adhere to the rules and maintain the decorum.

In-Lab Session Instructions

- Follow the instructions on the allotted exercises given in Lab Manual.
- Show the program and results to the instructors on completion of experiments.
- On receiving approval from the instructor, copy the program and results in the Lab record.
- Prescribed textbooks and class notes can be kept ready for reference if required.

THE STUDENTS SHOULD NOT

- Carry mobile phones while working with computer.
- Go out of the lab without permission.

LAB 1.1-LINUX BASIC COMMANDS

- Shell: a utility program that enables the user to interact with the Linux operating system. Commands entered by the user are passed by the shell to the operating system for execution. The results are then passed back by the shell and displayed on the user's display.
- Types of shell.

shell prompt

- a character at the start of the command line which indicates that the shell is ready to receive the commands.

Built-in commands

- echo
- echo $USER
- who
- whoami
- ls
- ls reg*
- ls -l

Built-in commands

- ls -l reg*
- ls –[options][file list][directory list]
- ls -a
- ls /
- pwd
- cd
- cd ..

Built-in commands

- cat
- cat > filename[Enter] type file content [Ctrl-D]
- cat filename
- cat filename1,filename2
- cp sourcefilename targetfilename
- cp filename(s) directoryname

BUILT-IN COMMANDS

- mv oldfilename newfilename
- rm filename
- mkdir directoryname
- rmdir directoryname
- Access Permission
- chmod
-

chmod

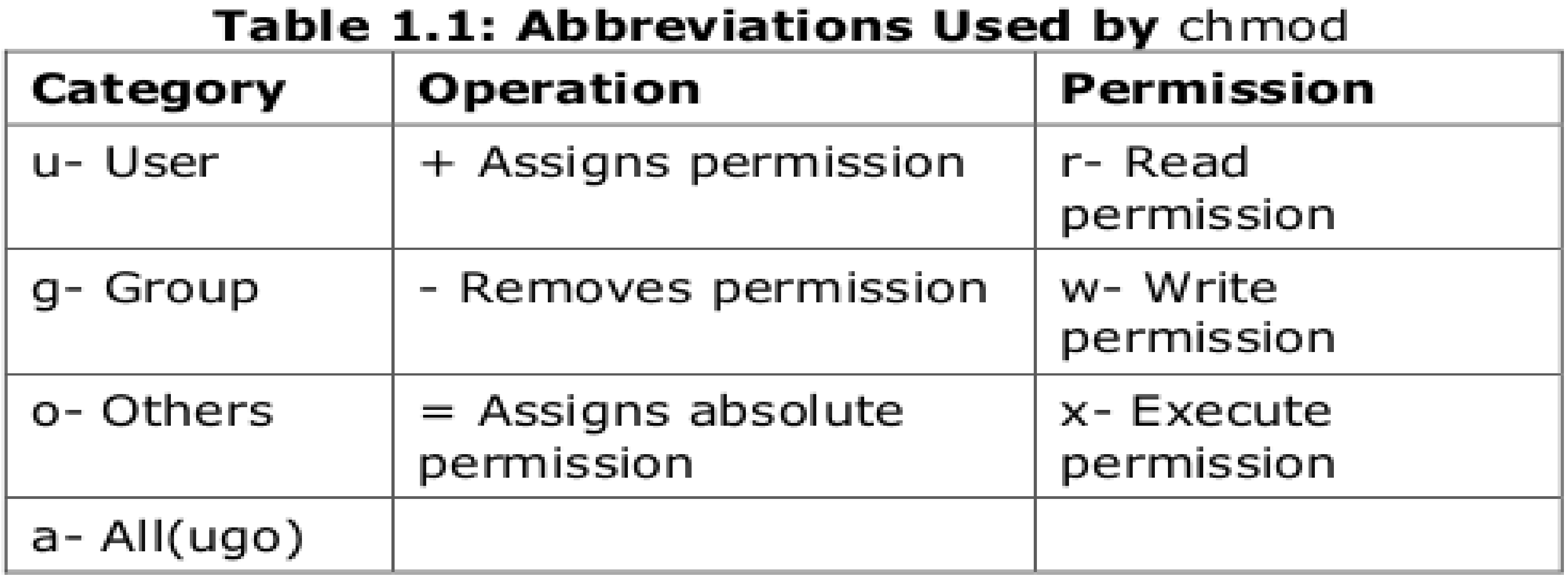


Table 1.1: Abbreviations Used by chmod

Category	**Operation**	**Permission**
u- User	+ Assigns permission	r- Read permission
g- Group	- Removes permission	w- Write permission
o- Others	= Assigns absolute permission	x- Execute permission
a- All(ugo)		

chmod

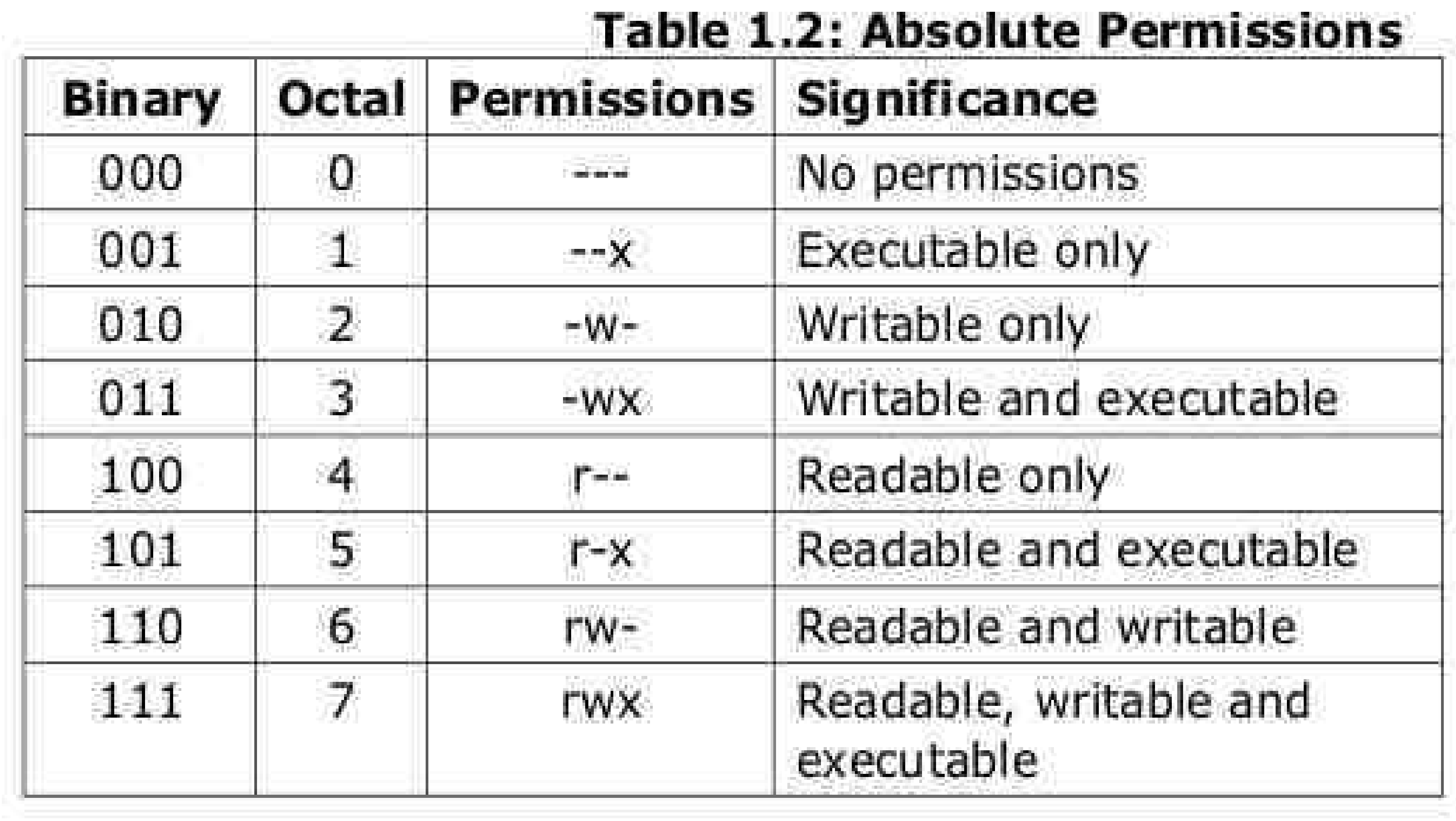


Table 1.2: Absolute Permissions

Binary	Octal	Permissions	Significance
000	0	---	No permissions
001	1	--x	Executable only
010	2	-w-	Writable only
011	3	-wx	Writable and executable
100	4	r--	Readable only
101	5	r-x	Readable and executable
110	6	rw-	Readable and writable
111	7	rwx	Readable, writable and executable

- **man {command name/system call name}**
- **wc**
-

Redirection operators

- > standard output operator
- < standard input operator
- cat <file1 >file2
- >> appending operator
- << document operator

LAB1.2 SHELL CONCEPTS

- **Wild-card: The metacharacters that are used to construct the generalized pattern for matching filenames belong to a category called wild-cards.**

Wild-card

- *: Any number of characters including none
- ?: A single or zero character
- [ijk]: A single character - either an i, j or k
- [x –z]: A single character between x and z
- [!ijk]:A single character that is not an i, j or k.
- [!x–z]:A single character not between x and z.
- {pat1, pat2, ….}:pat1, pat2, etc.

Pipes

- Standard input and standard output constitute two separate streams that can be individually manipulated by the shell. If so then one command can take input from the other. This is possible with the help of pipes.

Command substitution

- The shell enables the connecting of two commands in yet another way. While a pipe enables a command to obtain its standard input from the standard output of another command, the shell enables one or more command arguments to be obtained from the standard output of another command. This feature is called command substitution.
- echo The date today is `date`

Sequences

- **Two separate commands can be written in one line using “;” in sequences.**
- **Conditional Sequences**
  - cc hello.c && ./a.out
  - cc hello.c | | echo ‘Error’

LAB 1.3 File Filters commands in Linux

- head
- tail
- more
- grep < word name> < file name>
- sort
- sort -r

Reach Me
prakash.aithal@manipal.edu 9535829916

OSTL LAB 2

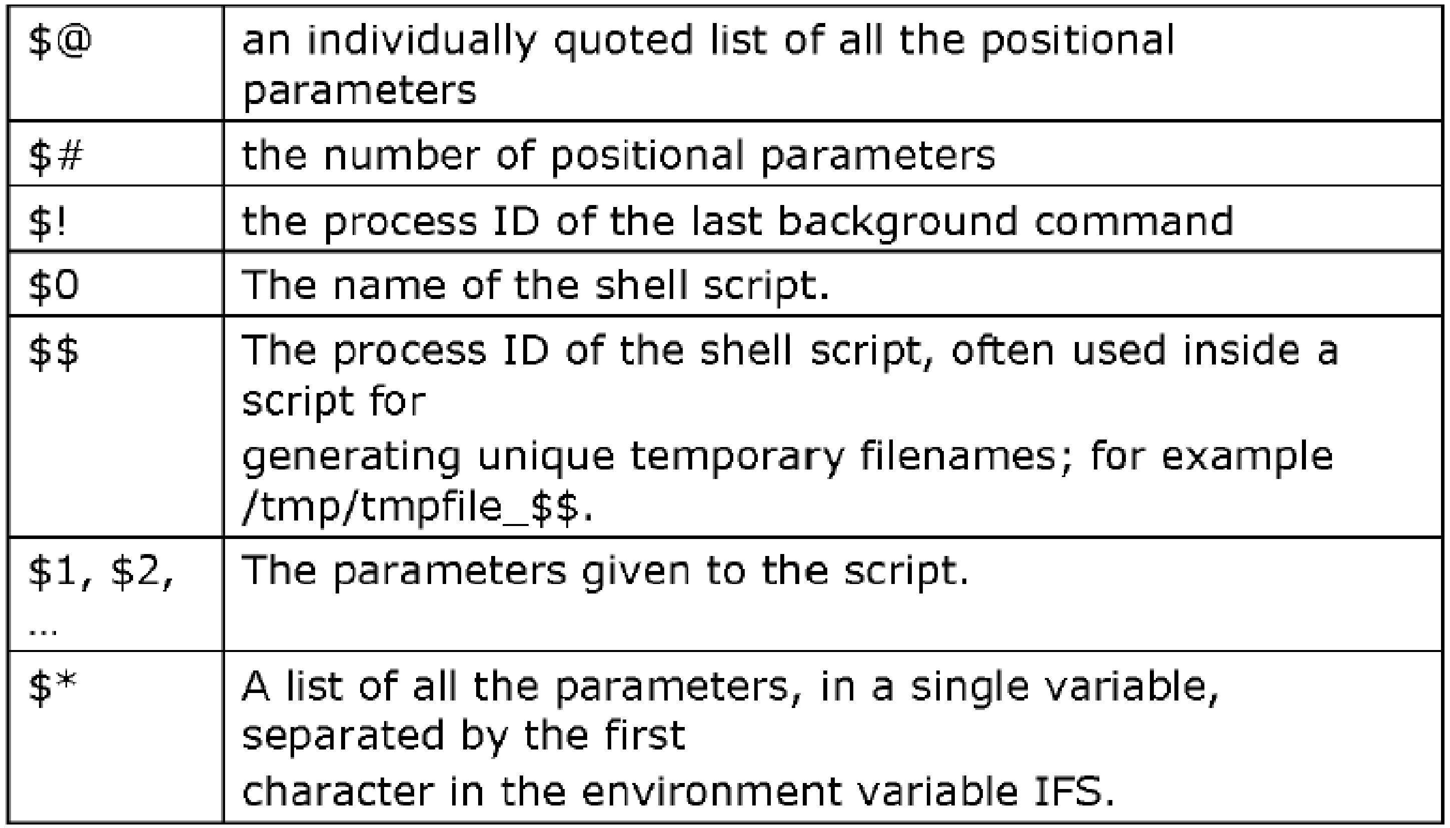
LAB 2- SHELL SCRIPTING I

2.1 : SHELL PROGRAMS - SCRIPTS

- A file that contains shell commands is called a script
- To run the script file, need to give execute permission to the file using chmod +x script
- To execute the script file, need to type its name

Variables used for script files

Table 2.1 :Parameter Variables



$@	an individually quoted list of all the positional parameters
$#	the number of positional parameters
$!	the process ID of the last background command
$0	The name of the shell script.
$$	The process ID of the shell script, often used inside a script for generating unique temporary filenames; for example /tmp/tmpfile_$$.
$1, $2, …	The parameters given to the script.
$*	A list of all the parameters, in a single variable, separated by the first character in the environment variable IFS.

Shell commands

- Print the following variables
  - $HOME
  - $PATH
  - $USER
  - $SHELL
  - $TERM
- Scope of Variables
  - Local variables are normal local variables limited to the shell which created it
    - To create local variable : - variableName=value
  - Environment variables are global variables e.g. $HOME, $USER, etc.
    - The export command is used to make a variable an environment variable

Operations on variables

- Simple assignment and access: variableName = value to assign, $variableName/ $(variableName) to access
- Testing of a variable for existence – print the variableName using echo
- Reading a variable from standard input – read variableName in the script file
- Making a variable read only – readonly {variable}
- Exporting a local variable to the environment – export variableName

Running jobs in Background

- Two ways to do this
  - & operator
  - The nohup command

Job Control

- The ps command : to list the processes
- The kill command: to kind one or more process
- The sleep command : to make the calling process sleep until specified number of seconds

OSTL LAB 3

LAB 3- SHELL SCRIPTING - II

3.1 : SHELL PROGRAMS - SCRIPTS

- Comments in script files start with # and go until the end of line
- Aliases- allows to define your own commands
  - Syntax: alias [word[=string]] e.g. alias dir =“ls –aF”
- Operators
  - Arithmetic
  - Relational
- Arithmetic operators
  - For arithmetic operations expr utility is used
  - E.g. x=1 then to increment it by 1 we write x=`expr $x +1` (Apostophe which is next to 1 in the keyboard)
  - To escape metacharacters like * which is wild card as well as multiplication operator using \

Test Expressions used for Relational Operators

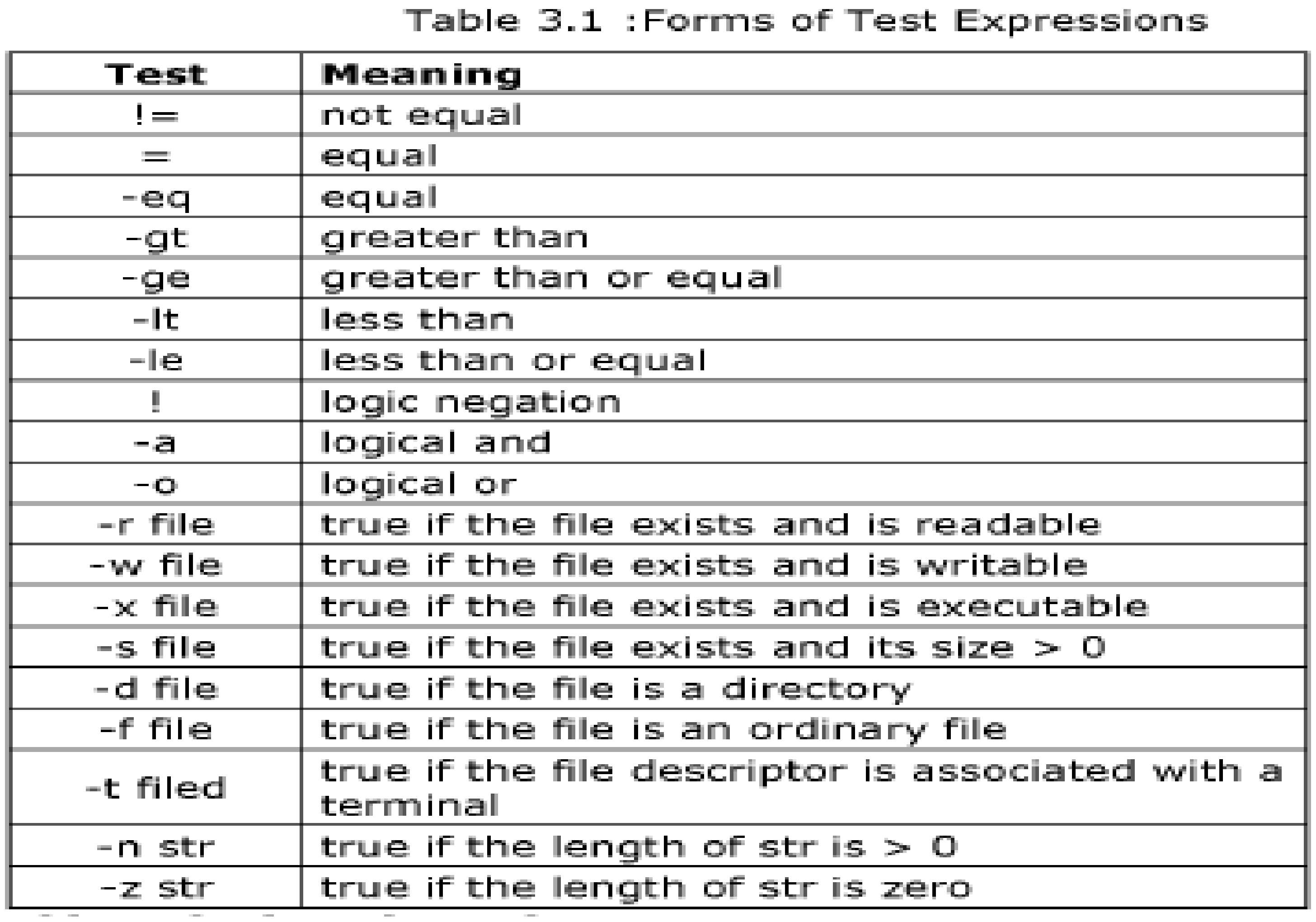


Table 3.1 :Forms of Test Expressions

Test	Meaning
!=	not equal
=	equal
-eq	equal
-gt	greater than
-ge	greater than or equal
-lt	less than
-le	less than or equal
!	logic negation
-a	logical and
-o	logical or
-r file	true if the file exists and is readable
-w file	true if the file exists and is writable
-x file	true if the file exists and is executable
-s file	true if the file exists and its size > 0
-d file	true if the file is a directory
-f file	true if the file is an ordinary file
-t filed	true if the file descriptor is associated with a terminal
-n str	true if the length of str is > 0
-z str	true if the length of str is zero

Control Structures

- The if condition
  - Syntax : if command 1

    ```
    then command 2
    fi
    ```
- The case condition
  - Syntax: case string in

    ```
    pattern1) commands1;;
    pattern2) commands2;;
    ……
    esac
    ```

Control Structures continued

- The while looping
  - Syntax: while condition is true

```
do
  commands
done
```

- The until looping
  - Syntax: until command-list1

```
do
  commands-list2
done
```

Control structures continued

- The for looping
  - Syntax: for variable in list

```
do
  command-list
done
```

OSTL LAB-4

compiled by,

Prakash Kalingrao Aithal

Compile c file in Linux

- gcc -o executablename filename.c

Run the executable

```
../executablefile
```

Header file in separate path

- gcc -I ../ filename.c -o executablefile

Include math library

- gcc -o mymath mymath.c -lm

Linking User Defined static object file

- gcc -o executablefile sourcefile.c staticobjectfile.o

ar command

- ar crv libraryfile.a objectfile1.o objectfile2.o .....

Linking library from different path

- gcc -o executablename sourcefilename -L pathname -llibraryname

Creating Shared Library

- gcc -fpic libraryfile.c
- gcc -shared -o libraryfile.so libraryfile.o
- Set the $LD_LIBRARY_PATH
- gcc -L PATH -o executablefile sourcefile.c -lfile

Makefile

macros

Target:Dependency

 action

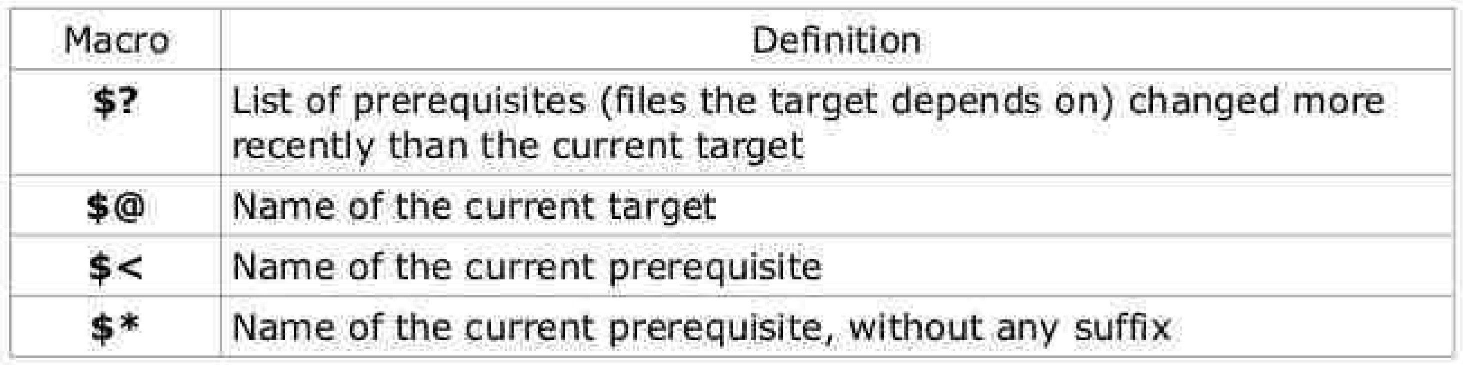
make

- make
- make makefile
- make Target
- make -f mymake

Macros

- CC=gcc
- INCLUDE=PATH

Macros



Macro	Definition
$?	List of prerequisites (files the target depends on) changed more recently than the current target
$@	Name of the current target
$<	Name of the current prerequisite
$*	Name of the current prerequisite, without any suffix

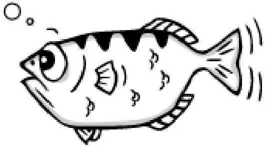
Macros

- `-`
- `@`

prakash.aithal@manipal.edu

9535829916

GDB: GNU Project Debugger



Open source Technologies

Content Courtesy:
Samuel Huang,
University of Maryland, USA

What is gdb?

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like *segmentation faults* may be easier to find with the help of gdb.
- https://www.gnu.org/software/gdb/documentation/ - online manual

What is gdb?

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like *segmentation faults* may be easier to find with the help of gdb.
- https://www.gnu.org/software/gdb/documentation/- online manual

What is gdb?

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like *segmentation faults* may be easier to find with the help of gdb.
- https://www.gnu.org/software/gdb/documentation/ - online manual

What is gdb?

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like *segmentation faults* may be easier to find with the help of gdb.
- https://www.gnu.org/software/gdb/documentation/ - online manual

What is gdb?

- “GNU Debugger”
- A debugger for several languages, including C and C++
- It allows you to inspect what the program is doing at a certain point during execution.
- Errors like segmentation faults may be easier to find with the help of gdb.
- https://www.gnu.org/software/gdb/documentation - online manual

Additional step when compiling

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

For example:

```
gcc –o fact fact.c
```

- Now you add a -goption to enable built-in debugging support (which gdbneeds):

```
gcc [other flags] -g <source files> -o <output file>
```

For example:

```
gcc -Wall -Werror -ansi -pedantic-errors -g prog1.c -o prog1.x
```

Additional step when compiling

- Normally, you would compile a program like:

```
gcc [flags] <source files> -o <output file>
```

For example:

```
gcc –o fact fact.c
```

- Now you add a –g option to enable built-in debugging support (which gdb needs):

```
gcc [other flags] -g <source files> -o <output file>
```

For example:

```
gcc –g –o fact fact.c
```

Starting up

Just try “gdb” or “gdb fact” You’ll get a prompt that looks like this:

```
(gdb)
```

If you didn’t specify a program to debug, you’ll have to load it in now:

```
(gdb) file prog1.x
```

Here, prog1.x is the program you want to load, and “file” is the command to load it.

Starting up

Just try “gdb” or “gdb fact ” You’ll get a prompt that looks like this:

```
(gdb)
```

If you didn’t specify a program to debug, you’ll have to load it in now:

```
(gdb) file fact
```

Here, fact is the executable of the program you want to load, and “file” is the command to load it.

Before we go any f th

gdb has an interactive shell, much like the one you use as soon as you log into the linux grace machines. It can recall history with the arrow keys, auto-complete words (most of the time) with the TAB key, and has other nice features.

Tip

If you're ever confused about a command or just want more information, use the "help" command, with or without an argument:

```
(gdb) help [command]
```

You should get a nice description and maybe some more useful tidbits…

Before we go any further

gdb has an interactive shell, much like the one you use as soon as you log into the linux grace machines. It can recall history with the arrow keys, auto-complete words (most of the time) with the TAB key, and has other nice features.

Tip

If you're ever confused about a command or just want more information, use the "help" command, with or without an argument:

```
(gdb) help [command]
```

You should get a nice description and maybe some more useful tidbits…

Running the program

To run the program, just use:

```
(gdb) run
```

This runs the program.

- If it has no serious problems (i.e. the normal program didn't get a segmentation fault, etc.), the program should run fine here too.
- If the program *did* have issues, then you (should) get some useful information like the line number where it crashed, and parameters to the function that caused the error:

Running the program

To run the program, just use:

```
(gdb) run
```

This runs the program.

- If it has no serious problems (i.e. the normal program didn't get a segmentation fault, etc.), the program should run fine here too.
- If the program *did* have issues, then you (should) get some useful information like the line number where it crashed, and parameters to the function that caused the error:

Running the program

To run the program, just use:

```
(gdb) run
```

This runs the program.

- If it has no serious problems (i.e. the normal program didn't get a segmentation fault, etc.), the program should run fine here too.
- If the program *did* have issues, then you (should) get some useful information like the line number where it crashed, and parameters to the function that caused the error:

So what if I have bugs?

Okay, so you've run it successfully. But you don't need gdb for that. What if the program *isn't* working?

Chances are if this is the case, you don't want to run the program without any stopping, breaking, etc. Otherwise, you'll just rush past the error and never find the root of the issue. So, you'll want to *step through* your code a bit at a time, until you arrive upon the error.

This brings us to the next set of commands…

So what if I have bugs?

Okay, so you've run it successfully. But you don't need gdb for that. What if the program *isn't* working?

Chances are if this is the case, you don't want to run the program without any stopping, breaking, etc. Otherwise, you'll just rush past the error and never find the root of the issue. So, you'll want to *step through* your code a bit at a time, until you arrive upon the error.

This brings us to the next set of commands…

So what if I have bugs?

Okay, so you've run it successfully. But you don't need gdb for that. What if the program *isn't* working?

Chances are if this is the case, you don't want to run the program without any stopping, breaking, etc. Otherwise, you'll just rush past the error and never find the root of the issue. So, you'll want to *step through* your code a bit at a time, until you arrive upon the error.

This brings us to the next set of commands…

Understand the commands

Run (r)	r *file* start your program
Break (b)	b *5* or b *fn* set breakpoint at *line* or *fn*
Delete (d)	d *5* delete the breakpoint at line *n*
Continue (c)	c continue running
Next (n)	n execute next line, including any function calls
Step (s)	s execute until another line reached
Print (p)	p *var* display the value of *var*

Setting breakpoints

Breakpoints can be used to stop the program run in the middle, at a designated point. The simplest way is the command "break." This sets a breakpoint at a specified file-line pair:

```
(gdb) break file1.c:6
```

This sets a breakpoint at line 6, of file1.c. Now, **if** the program ever reaches that location when running, the program will pause and prompt you for another command.

Tip

You can set as many breakpoints as you want, and the program should stop execution if it reaches any of them.

More learning with breakpoints

You can also tell gdb to break at a particular function. Suppose you have a function my_func:

```
int my_func(int a, char*b);
```

You can break anytime this function is called:

```
(gdb) break my_func
```

Now what?

- Once you've set a breakpoint, you can try using the run command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).
- You can proceed onto the next breakpoint by typing "continue" (Typing run again would restart the program from the beginning, which isn't very useful.)

```
(gdb) continue
```

- You can single-step (execute *just* the next line of code) by typing "step." This gives you really fine-grained control over how the program proceeds. You can do this a *lot*...

```
(gdb) step
```

Now what?

- Once you've set a breakpoint, you can try using the run command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).
- You can proceed onto the next breakpoint by typing "continue" (Typing run again would restart the program from the beginning, which isn't very useful.)

```
(gdb) continue
```

- You can single-step (execute *just* the next line of code) by typing "step." This gives you really fine-grained control over how the program proceeds. You can do this a *lot*...

```
(gdb) step
```

Now what?

- Once you've set a breakpoint, you can try using the run command again. This time, it should stop where you tell it to (unless a fatal error occurs before reaching that point).
- You can proceed onto the next breakpoint by typing "continue" (Typing run again would restart the program from the beginning, which isn't very useful.)

```
(gdb) continue
```

- You can single-step (execute *just* the next line of code) by typing "step." This gives you really fine-grained control over how the program proceeds. You can do this a *lot*...

```
(gdb) step
```

Now what? (even more!)

- Similar to “step,” the “next” command single-steps as well, except this one doesn’t execute each line of a sub-routine, it just treats it as one instruction.

```
(gdb) next
```

Tip

Typing “step” or “next” a lot of times can be tedious. If you just press ENTER, gdb will repeat the same command you just gave it. You can do this a bunch of times.

Now what? (even more!)

- Similar to "step," the "next" command single-steps as well, except this one doesn't execute each line of a sub-routine, it just treats it as one instruction.

```
(gdb) next
```

Tip

Typing "step" or "next" a lot of times can be tedious. If you just press ENTER, gdb will repeat the same command you just gave it. You can do this a bunch of times.

Querying other aspects

- So far you've learned how to interrupt program flow at fixed, specified points, and how to continue stepping line-by-line.
  However, sooner or later you're going to want to see things like *the values of variables*, etc. This *might* be useful in debugging.
- The print command prints the value of the variable specified, and print/x prints the value in hexadecimal:

```
(gdb) print my_var
(gdb) print/x my_var
```

Other useful commands

- backtrace - produces a stack trace of the function calls that lead to a seg fault (should remind you of Java exceptions)
- where - same as backtrace; you can think of this version as working even when you're still in the middle of the program
- finish - runs until the current function is finished
- delete - deletes a specified breakpoint
- info breakpoints - shows information about all declared breakpoints

More about breakpoints

Breakpoints by themselves may seem too tedious. You have to keep stepping, and stepping, and stepping…

Once we develop an idea for what the error could be (like dereferencing a NULL pointer, or going past the bounds of an array), we probably only care if such an event happens; we don't want to break at each iteration regardless.

So ideally, we'd like to *condition* on a particular requirement (or set of requirements). Using **conditional** breakpoints allow us to accomplish this goal…

Thank you ☺

git --everything-is-local

Git for Version Control

Open source Technologies

Content Courtesy:
Ruth E. Anderson
Department of Computer Science & Engineering
University of Washington

Version control systems

- ✓ Version control (or revision control, or source control) is all about managing multiple versions of documents, programs, web sites, etc.
  - Almost all "real" projects use some kind of version control
  - Essential for team projects, but also very useful for individual projects
- ✓ Some well-known version control systems are CVS, Subversion, Mercurial, and Git
  - CVS and Subversion use a "central" repository; users "check out" files, work on them, and "check them in"
  - Mercurial and Git treat all repositories as equal
- ✓ Distributed systems like Mercurial and Git are newer and are gradually replacing centralized systems like CVS and Subversion

Why version control?

For working by yourself:

- Gives you a “time machine” for going back to earlier versions
- Gives you great support for different versions (standalone, web app, etc.) of the same basic project

For working with others:

- Greatly simplifies concurrent work, merging changes

About Git

git --everything-is-local

- Created by Linus Torvalds, creator of Linux, in 2005
  - Came out of Linux development community
  - Designed to do version control on Linux kernel
- Goals of Git:
  - Speed
  - Support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects efficiently

  - *(A "git" is a cranky old man. Linus meant himself.)*

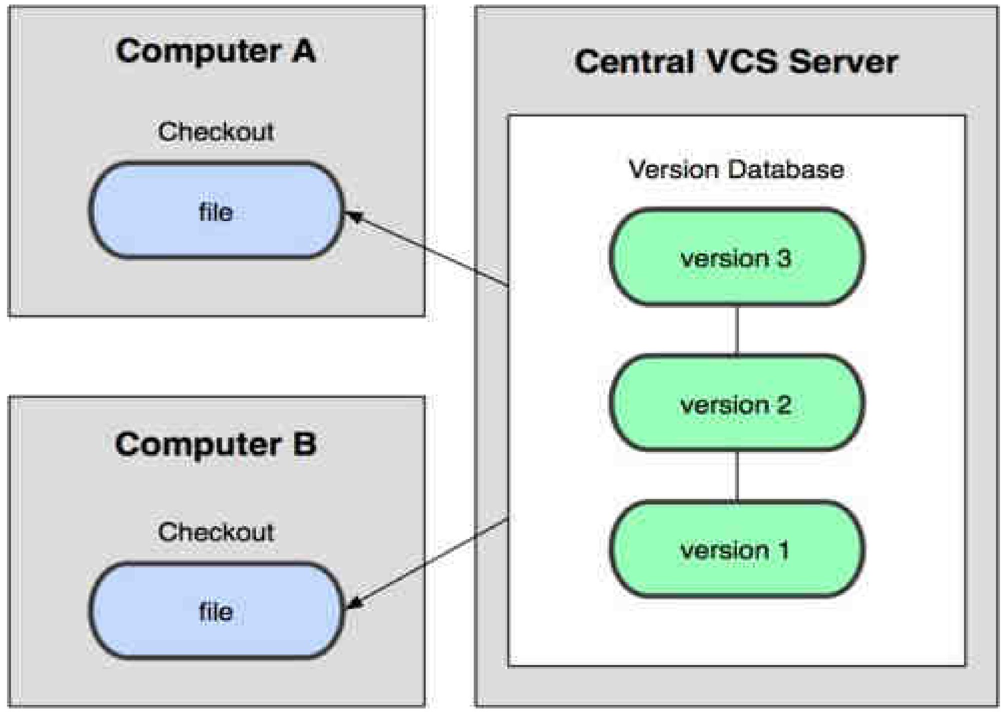
Installing/learning Git

- Git website: http://git-scm.com/
  - Free on-line book: http://git-scm.com/book
  - Reference page for Git: http://gitref.org/index.html
  - Git tutorial: https://www.atlassian.com/git/tutorials
  - Git for Computer Scientists:
    - http://eagain.net/articles/git-for-computer-scientists/

- At command line: *(where verb = config, add, commit, etc.)*
  - `git help verb`

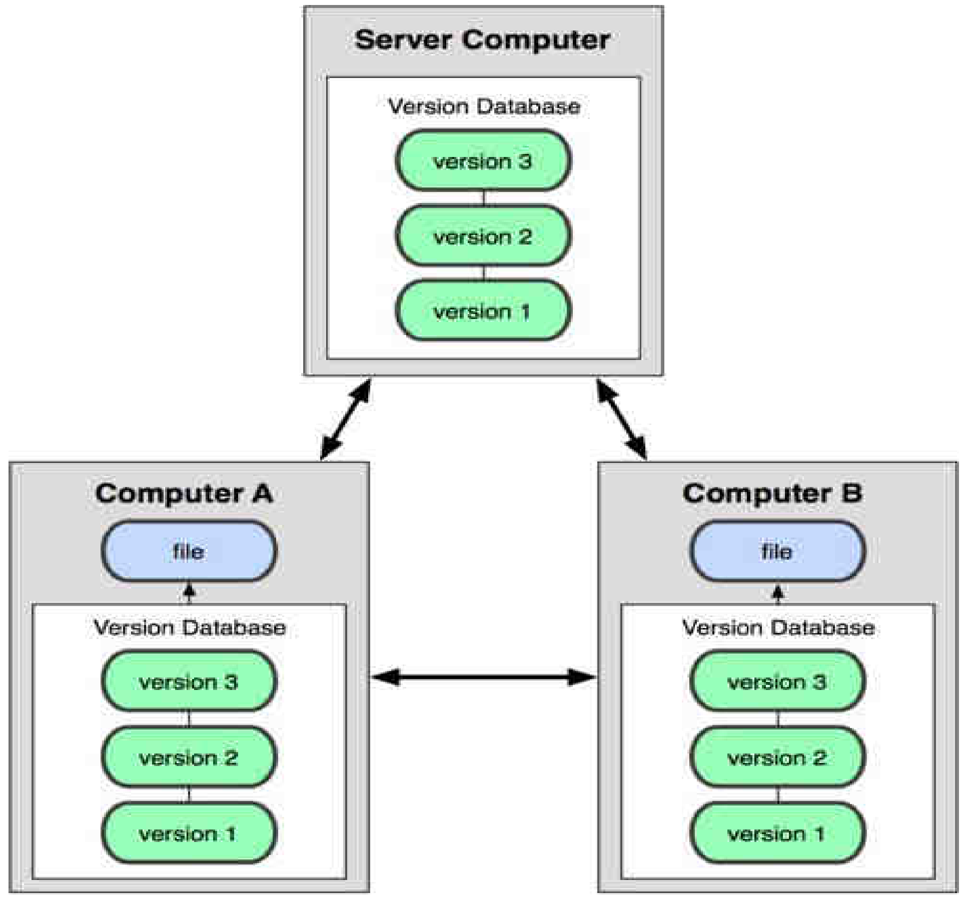
Centralized VCS

- In Subversion, CVS, Perforce, etc. A central server repository (repo) holds the "official copy" of the code
  - the server maintains the sole version history of the repo
- You make "checkouts" of it to your local copy
  - you make local modifications
  - your changes are not versioned
- When you're done, you "check in" back to the server
  - your checkin increments the repo's version



Distributed VCS (Git)

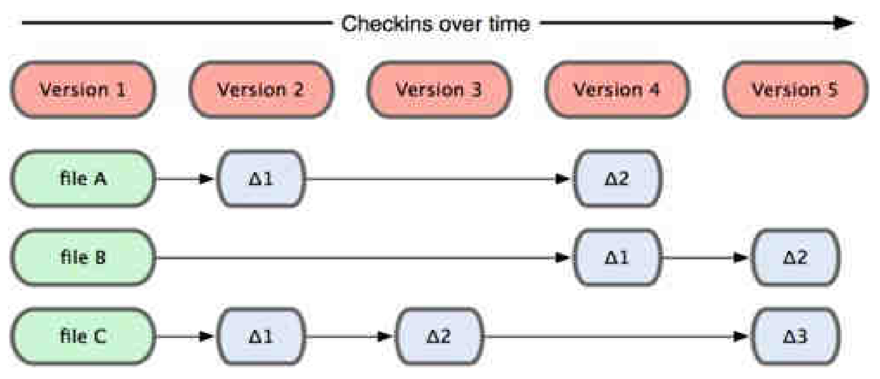
- In **git**, mercurial, etc., you don't "checkout" from a central repo
  - you "clone" it and "pull" changes from it
- Your local repo is a complete copy of everything on the remote server
  - yours is "just as good" as theirs
- Many operations are local:
  - check in/out from *local* repo
  - commit changes to *local* repo
  - local repo keeps version history
- When you're ready, you can "push" changes back to server



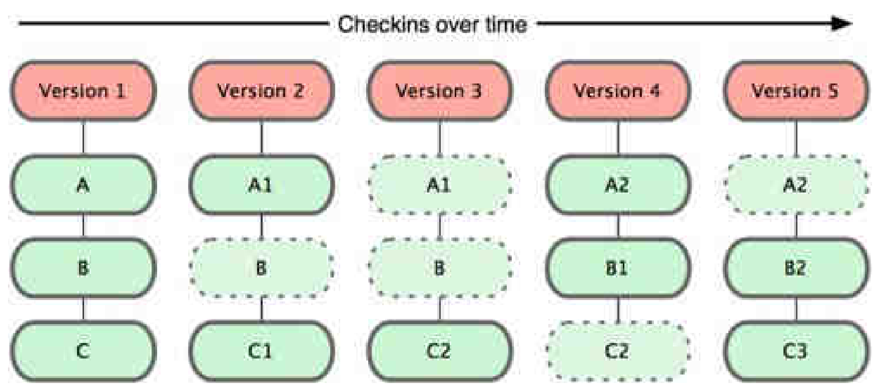
Git snapshots

- Centralized VCS like Subversion track version data on each individual file.
- Git keeps "snapshots" of the entire state of the project.
  - Each checkin version of the overall code has a copy of each file in it.
  - Some files change on a given checkin, some do not.
  - More redundancy, but faster.

Subversion

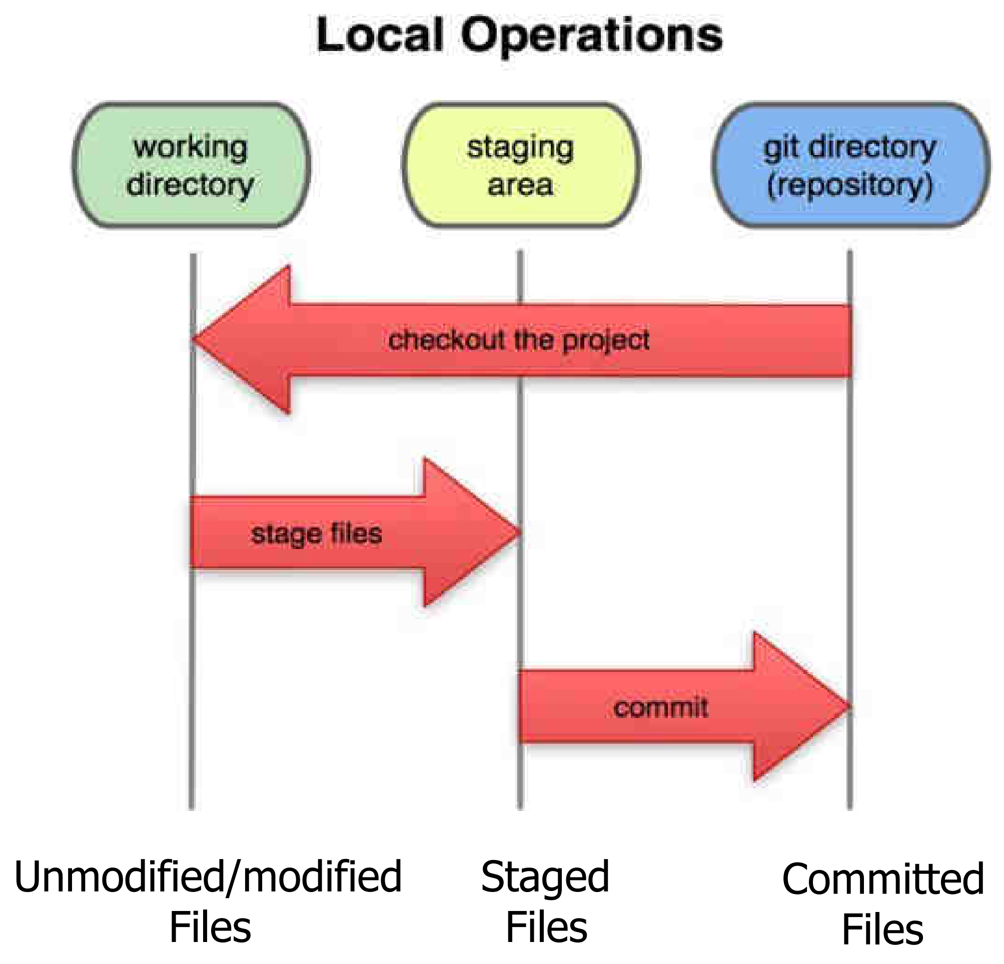


Git



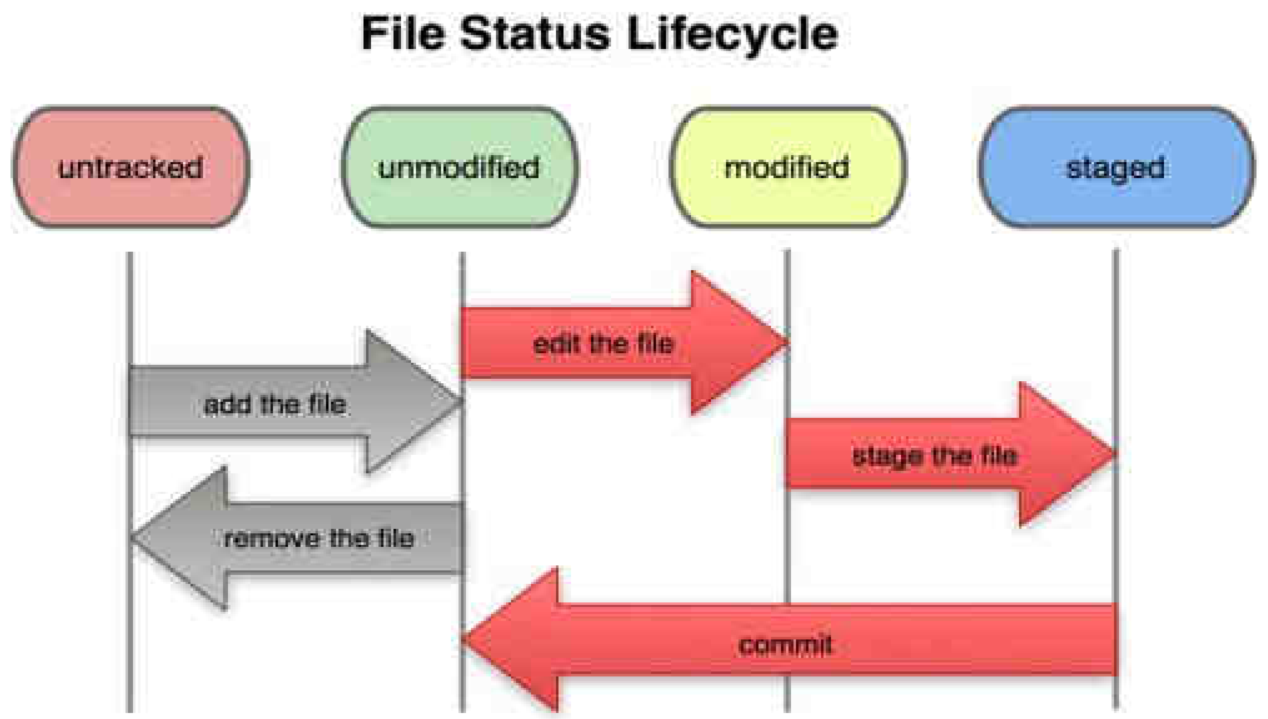
Local git areas

- In your local copy on git, files can be:
  - In your local repo
    - (committed)
  - Checked out and modified, but not yet committed
    - (working copy)
  - Or, in-between, in a **"staging" area**
    - Staged files are ready to be committed.
    - A commit saves a snapshot of all staged state.



Basic Git workflow

- **Modify** files in your working directory.
- **Stage** files, adding snapshots of them to your staging area.
- **Commit**, which takes the files in the staging area and stores that snapshot permanently to your Git directory.



Git commit checksums

- In Git, each user has their own copy of the repo, and commits changes to their local copy of the repo before pushing to the central server.
- So Git generates a unique **SHA-1 hash** (40 character string of hex digits) for every commit.
- Refers to commits by this ID rather than a version number.
- Often we only see the first 7 characters:
  - `1677b2d Edited first line of readme`
  - `258efa7 Added line to readme`
  - `0e52da7 Initial commit`

Initial Git configuration

- Check whether git is installed
  - `git --version`
- Set the name and email for Git to use when you commit:
  - `git config --global user.name "raj293"`
  - `git config --global user.email raj293@yahoo.com`
  - You can call `git config –list` to verify these are set.
- Set the editor that is used for writing commit messages:
  - `git config --global core.editor nano or gedit`
    - (it may be vim by default)

Creating a Git repo

Two common scenarios

- To create a new **local Git repo** in your current directory:
  - `git init`
    - This will create a `.git` directory in your current directory.
    - Then you can commit files in that directory into the repo.
  - `git add filename`
  - `git commit -m "commit message"`
- To **clone a remote repo** to your current directory:
  - `git clone url      localDirectoryName`
    - This will create the given local directory, containing a working copy of the files from the repo, and a `.git` directory (used to hold the staging area and your actual local repo)

Git commands

command	description
git clone ***url [dir]***	copy a Git repository so you can add to it
git add ***file***	adds file contents to the staging area
git commit	records a snapshot of the staging area
git status	view the status of your files in the working directory and staging area
git diff	shows diff of what is staged and what is modified but unstaged
git help [***command***]	get help info about a particular command
git pull	fetch from a remote repo and try to merge into the current branch
git push	push your new branches and data to a remote repository
others: init, reset, branch, checkout, merge, log, tag	

Add and commit a file

- The first time we ask a file to be tracked, *and every time before we commit a file*, we must add it to the staging area:
  - `git add f1.txt f2.txt`
    - Takes a snapshot of these files, adds them to the staging area.
    - In Git, "add" means "add to staging area" so it will be part of the next commit.

- To move staged changes into the repo, we commit:
  - `git commit -m "Fixing bug #22"`

- To undo changes on a file before you have committed it:
  - `git reset HEAD -- filename` (unstages the file)
  - `git checkout -- filename` (undoes your changes)
  - All these commands are acting on your local version of repo.

Viewing/undoing changes

- To view status of files in working directory and staging area:
  - `git status` or `git status -s` (short version)
- To see what is modified but unstaged:
  - `git diff`
- To see a list of staged changes:
  - `git diff --cached`
- To see a log of all changes in your local repo:
  - `git log` or `git log --oneline` (shorter version)

```
1677b2d Edited first line of readme  258efa7 Added line to
readme  0e52da7 Initial commit
```

    - `git log -5` (to show only the 5 most recent updates), etc.

Branching and merging

Git uses branching heavily to switch between multiple tasks.

- To create a new local branch:
  - `git branch name`
- To list all local branches: (* = current branch)
  - `git branch`
- To switch to a given local branch:
  - `git checkout branchname`
- To merge changes from a branch into the local master:
  - `git checkout master`
  - `git merge branchname`

Interaction with remote repository

- **Push** your local changes to the remote repo.
- **Fetch** from remote repo to get most recent changes. (meta data)
- **Pull** from remote repo to get most recent changes. (all data)
  - (fix conflicts if necessary, add/commit them to your local repo)
- To fetch the most recent updates from the remote repo into your local repo, and put them into your working directory:
  - `git pull origin master`
- To put your changes from your local repo in the remote repo:
  - `git push origin master`

GitHub

- GitHub.com is a site for online storage of Git repositories.
  - You can create a **remote repo** there and push code to it.
  - Many open source projects use it, such as the Linux kernel.
  - You can get free space for open source projects, or you can pay for private projects.
    - Free private repos for educational use: github.com/edu

- *Question:* Do I always have to use GitHub to use Git?
  - *Answer:* No! You can use Git locally for your own purposes.
  - Or you or someone else could set up a server to share files.
  - Or you could share a repo with users on the same file system, as long everyone has the needed file permissions).

Git clone command

- To **clone a remote repo** to your current directory:
  - `git clone` *`url localDirectoryName`*
    - This will create the given local directory, containing a working copy of the files from the repo, and a `.git` directory (used to hold the staging area and your actual local repo)

Thank you

☺

HTML

Open source Technologies Lab

HTML5 – Overview

- HTML5 is the next major revision of the HTML standard superseding HTML 4.01, XHTML 1.0, and XHTML 1.1. HTML5 is a standard for structuring and presenting content on the World Wide Web.
- HTML5 is a cooperation between the World Wide Web Consortium (W3C) and the Web Hypertext Application Technology Working Group (WHATWG).
- The new standard incorporates features like video playback and drag-and-drop that have been previously dependent on third-party browser plug-ins such as Adobe Flash, Microsoft Silverlight, and Google Gears.

New Features

HTML5 introduces a number of new elements and attributes that can help you in building modern websites. Here is a set of some of the most prominent features introduced in HTML5.

- **New Semantic Elements:** These are like <header>, <footer>, and <section>.
- **Forms 2.0:** Improvements to HTML web forms where new attributes have been introduced for <input> tag.
- **Canvas:** This supports a two-dimensional drawing surface that you can program with JavaScript.
- **Audio & Video:** You can embed audio or video on your webpages without resorting to third-party plugins.
- **WebSocket :** A next-generation bidirectional communication technology for web applications.
- **Server-Sent Events:** HTML5 introduces events which flow from web server to the web browsers and they are called Server-Sent Events (SSE).
- **Geolocation:** Now visitors can choose to share their physical location with your web application.
- **Microdata:** This lets you create your own vocabularies beyond HTML5 and extend your web pages with custom semantics.
- **Drag and drop:** Drag and drop the items from one location to another location on the same webpage.

Backward Compatibility

- HTML5 is designed, as much as possible, to be backward compatible with existing web browsers. Its new features have been built on existing features and allow you to provide fallback content for older browsers.
- It is suggested to detect support for individual HTML5 features using a few lines of JavaScript.

HTML Syntax

The DOCTYPE

- DOCTYPEs in older versions of HTML were longer because the HTML language was SGML based and therefore required a reference to a DTD.
- HTML 5 authors would use simple syntax to specify DOCTYPE as follows:

```
<!DOCTYPE html>
```

- The above syntax is case-insensitive.

HTML Syntax

Character Encoding

- HTML 5 authors can use simple syntax to specify Character Encoding as follows:

```
<meta charset="UTF-8">
```

- The above syntax is case-insensitive.

HTML Syntax

The <script> tag

- It's common practice to add a type attribute with a value of "text/javascript" to script elements as follows

```
<script type="text/javascript" src="scriptfile.js"></script>
```

- HTML 5 removes extra information required and you can use simply following syntax

```
<script src="scriptfile.js"></script>
```

HTML Syntax

The <link> tag

- So far you were writing <link> as follows

```
<link rel="stylesheet" type="text/css" href="stylefile.css">
```

- HTML 5 removes extra information required and you can simply use the following syntax

```
<link rel="stylesheet" href="stylefile.css">
```

HTML5 Elements

- HTML5 elements are marked up using start tags and end tags. Tags are delimited using angle brackets with the tag name in between.
- The difference between start tags and end tags is that the latter includes a slash before the tag name.
- Following is the example of an HTML5 element

```
<p>...</p>
```

- HTML5 tag names are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.
- Most of the elements contain some content like <p>...</p> contains a paragraph. Some elements, however, are forbidden from containing any content at all and these are known as void elements. For example, **br**, **hr**, **link**, **meta**, etc.
- A complete list of HTML5 Elements (https://www.tutorialspoint.com/html5/html5_tags.htm)

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

HTML5 Attributes

- Elements may contain attributes that are used to set various properties of an element.
- Some attributes are defined globally and can be used on any element, while others are defined for specific elements only. All attributes have a name and a value and look like as shown below in the example.
- Following is the example of an HTML5 attribute which illustrates how to mark up a **div** element with an attribute named class using a value of "example"

```
<div class="example">...</div>
```

- Attributes may only be specified within start tags and must never be used in end tags.
- HTML5 attributes are case insensitive and may be written in all uppercase or mixed case, although the most common convention is to stick with lowercase.
- Here is a complete list of HTML5 Attributes (https://www.tutorialspoint.com/html5/html5_attributes.htm).

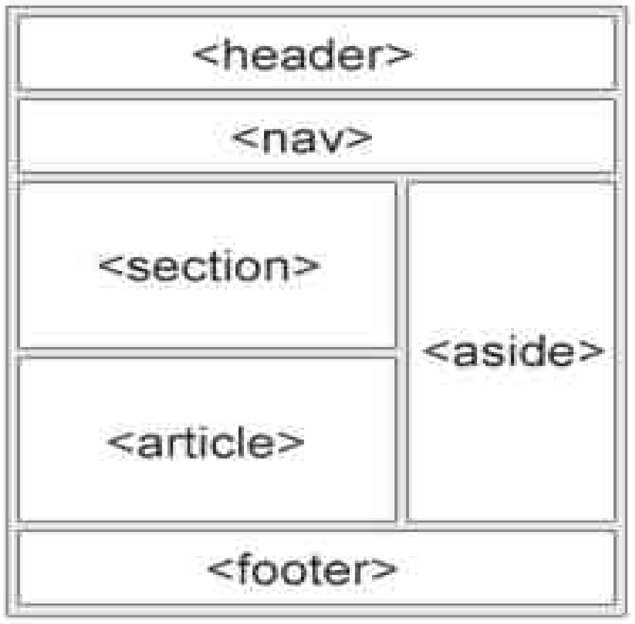
HTML5 Document

The following tags have been introduced for better structure –

- **section:** This tag represents a generic document or application section. It can be used together with h1-h6 to indicate the document structure.
- **article:** This tag represents an independent piece of content of a document, such as a blog entry or newspaper article.
- **aside:** This tag represents a piece of content that is only slightly related to the rest of the page.
- **header:** This tag represents the header of a section.
- **footer:** This tag represents a footer for a section and can contain information about the author, copyright information, etcetera.
- **nav:** This tag represents a section of the document intended for navigation.
- **dialog:** This tag can be used to mark up a conversation.
- **figure:** This tag can be used to associate a caption together with some embedded content, such as a graphic or video.

HTML5 Document

- There is no concept of validation in HTML5; because HTML5 is not defined as SGML or XML application.
  - <article>
  - <aside>
  - <details>
  - <figcaption>
  - <figure>
  - <footer>
  - <header>
  - <main>
  - <mark>
  - <nav>
  - <section>
  - <summary>
  - <time>



```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "utf-8">
        <title>...</title>
    </head>

    <body>
        <header>...</header>
        <nav>...</nav>

        <article>
            <section>
                ...
            </section>
        </article>
        <aside>...</aside>

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

Web Forms 2.0

- Web Forms 2.0 is an extension to the forms features found in HTML4. Form elements and
- attributes in HTML5 provide a greater degree of semantic mark-up than HTML4 and free us from a great deal of tedious scripting and styling that was required in HTML4.

HTML5 – Canvas

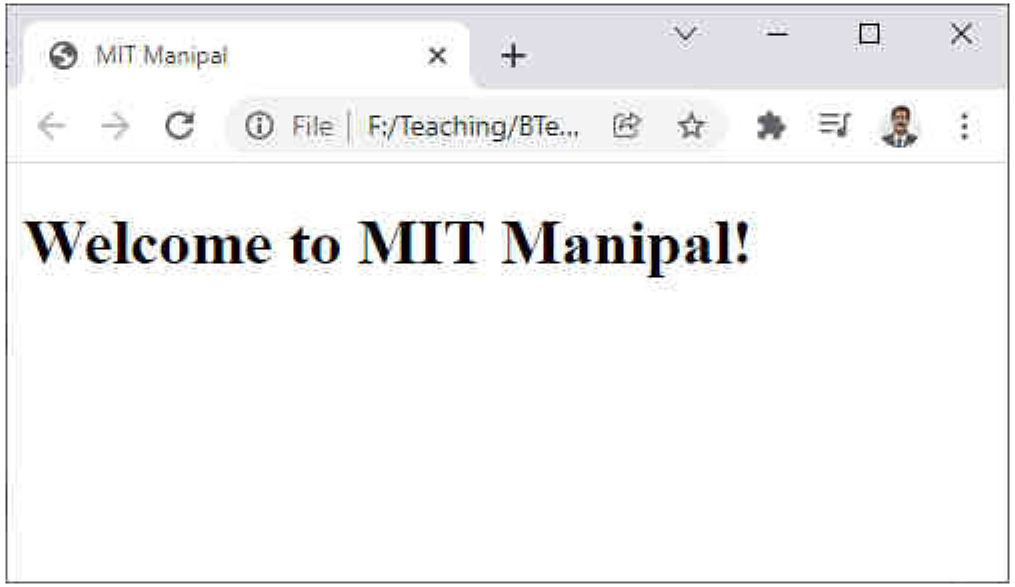
- HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript.
- It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.

Topics

1. HTML5
2. Elements, Tags, Attributes
3. Formatting Tags
4. Styles
5. Lists
6. Tables
7. Embedding Images
8. Embedding Media (Audio and Video)
9. HTML Elements & Layouts
10. Web forms
11. Canvas

First HTML program

- Structure of an HTML document
- Write a sample HTML program
- Save and run the HTML program



Welcome to MIT Manipal!

Element, Tags and Attributes

- HTML document contains a set of elements
  - Each element consists of some content between tags
  - Some elements don't require any content - Empty Elements
  - We can also have Nested Elements
- The contents to be structured on the web are defined by Tags
  - Written in between the < > (angular brackets)
  - Tags will have a start and an end

    syntax: <tag-name> content </tag-name>
  - End tag contains a / (forward slash) before the tag name
  - Some tags don't require an end tag <br/>
  - HTML Tags are not case sensitive
  - It is advisable to use lower-case

Element, Tags and Attributes

- Attributes are additional information pertaining to an Element
  - Written inside the starting tag
  - Declared by name - value pair separated by = symbol

  syntax: <tag-name attr-name=attr-value> content </tag-name>

- Values containing spaces have to be entered within single or double quotes

  <tag-name attr-name="attr value"> content </tag-name>

- Multiple attributes can be specified with help of ; (semicolon) separator

  <tag-name attr-name1="attr value"; attr-name2="attr value"> content </tag-name>

Some Common Attributes

- title
- href
- id
- class
- Style
- Values that contain more than one word, can be declared within single or double quotes. It is mandatory to close the quotes, else the attribute won't be applied.

Formatting Tags

- HTML allows us to do basic formatting of the content
- Formatting tags are exactly same as in any word processor application

- Headings (h1, h2, h3, h4, h5, h6)
- Paragraph
- Bold, Italic, Underline
- Large and Small
- Insert and Delete
- Superscript and Subscript
- Line Break
- Horizontal Ruler
- Anchor

HTML Headings

- In any document, we use headings for various purposes in various sizes
- HTML allows us to create headings with six different sizes
- Heading tag will start from h1 for large headings and ends with h6 for smaller ones
- A line space will be added automatically before and after the headings

Break Tags

- <br/> or <br>
- Break tag is a self-closing tag which doesn't require an end tag

Anchor Tag

- In our webpage we could have many links that navigate us to either:
  - a section in the same page
  - a different page
  - an external website
- This action can be created with the help of <a> tag
- This tag is also called as hyperlink tag
- Anchor tag should always be declared along with its attribute – href
- href stands for hyperlink reference
- The value of href holds the target link

Anchor Tag

Syntax: `<a href="target link"> content </a>`

- The content within the Anchor tag is clickable
- By clicking on this text, we will be redirected to the targeted link
- The content can be a text or an image

[For redirecting link make sure that the specified HTML file is available in the same folder or with full path]

Anchor Tag - target

- target is an another important attribute for the anchor tag
- It species where the target has to be redirected
- The values are
  - _blank: Opens in a new tab
  - _parent: Opens in the parent frame
  - _self: Opens in the same frame
  - _top: Opens in the full body of the window

Styles

- Allows us to define the presentation of the HTML Elements
- Defined in 3 ways:
  - Inline attribute for the tags
  - Individual tag
  - External style sheet (CSS)

Style Attribute

- Style is an attribute that specifies the style of an element
  - For eg: font-family, font-size, colour, etc.
  - Style attribute decides how the content will be displayed
  - Once the style attribute is declared for a tag, it will override the existing property

Syntax: <tag-name style = "Property-Name:Value"> content </tag-name>

- We can add more properties with the help of semi-colon

<tag-name style = "Property1:Value; Property2:Value"> content </tag-name>

Style Tag

- It defines the styles for the whole document
- We can have many style tags for a single document
- It has to be declared inside the head section
- It has both start and an end tags

    syntax: <style> tag-name { property:value; } </style>

- We can add styles for multiple tags
- Multiple attributes for the tags

Common Style Properties

- background-color to set the background
  style="background-color:red;"
- color to set the font color
  style="color:blue;"
- font-family to set the font
  style="font-family:courier;“
- font-size to set the font text size
  style="font-size:24px;"
- text-align for text alignment
  style="text-align:right;"

CSS

- External styles are defined with the help of CSS
- CSS - Cascading Style Sheet is a Single file, which can define the styles for multiple HTML pages
- CSS file will not have any HTML code
- Include the CSS file in the head section with the help of link tag

    syntax: `<link rel = "stylesheet" href = "filename.css">`

Lists

- Groups the related content together
- Create structured navigations like menu bars

Types of List

- Unordered List
- Ordered List
- Definition or Description List
- All these types have a start and an end tag

syntax:

```
<type of list>
<li> list item 1 </li>
<li> list item 2 </li>
.
.
<li> list item n </li>
</type of list>
```

UL Style Types

- We can use CSS and style to modify how the Unordered List should be displayed
- We have to specify the style inside the start tag

```
<ul style = "list-style-type:style">
```

  - Disc (default)
  - Circle
  - Square
  - None

OL Types

- Numbers (1, 2, 3)
- Upper Case Letters (A, B, C)
- Lower Case Letters (a, b, c)
- Roman Numerals in Uppercase (I, II, III)
- Roman Numerals in Lowercase (i, ii, iii)

Description List

- Description List <dl> is not like the other two lists
- It will display the list in the form of a name - value pair
- That means it will display the line item without prefixing it with a bullet or a number
- Each list item will have a name and a value to it.
  - Instead of li tag, we have to use dt and dd tags
  - ❖ dt - Term or name
  - ❖ dd - Term definition or value
  - ❖ dt and dd have a start and end tag

Description List

- Must have at least one term and a value assigned to it
- We can have multiple values for a single term

```
<dl>
<dt> Name 1 </dt>
<dd> value 1 </dd>
<dt> Name 2 </dt>
<dd> value 1 </dd>
<dd> value 2 </dd>
</dl>
```

Nested List

- We can also have a list inside a list. This is called a Nested List
- Both the lists need not be the same
  - For eg: Both need not be ordered or unordered or descriptive

```
<ol>
    <li> Unit 1 </li>
    <ul>
        <li> Chapter 1 </li>
        <li> Chapter 2 </li>
    </ul>
    <li> Unit 2 </li>
    <ul>
        <li> Chapter 1 </li>
        <li> Chapter 2 </li>
    </ul>
</ol>
```

```
1. Unit 1
     • Chapter 1
     • Chapter 2
2. Unit 2
     •  Chapter 1
     •  Chapter 2
```

Tables

- Tables are used to format the content to be displayed on the web, in the form of Rows and Columns

Table Attributes

- HTML attributes and styles
  - Border
  - Width
  - Spacing and Padding
  - Rowspan and Columnspan

Table Attributes

- The Rowspan and Columnspan are declared inside the td and th tag
- The others are declared inside the table start tag or in a style tag or attribute

Spacing and Padding

- Border Spacing - Space between the cells
- Padding - Space between the cell content and cell border

Comments

- Helps us to understand the code
- It enables us to write documentation of a program inside the code itself
- Helps us to debug the code
- It also helps us to skip a snippet of code without executing it

syntax: `<!-- our comments -->`

HTML Entities

- In HTML some characters are reserved
- The browser will consider those characters as a part of HTML syntax
- We can use these reserved characters in the content in the form of HTML Entity
- The name or number of the entities have to be written in between ampersand and semicolon symbols
- Blank space ( ) - &nbsp;
- Less than (<) - &lt;
- Greater than (>) - &gt;
- ampersand (&) - &amp;

Embedding Images

In HTML, we can embed the image as a

- Normal image
- Background image
- Link
- Single image with multiple links

syntax: <img src="image name along with the path">

- Before embedding any media, always ensure that you have all the rights to use the file
- Sharing files on the web without reusable permission, is illegal

Embedding Images

We can embed an image in any one of the following ways

- From the computer
- Downloading from the internet
- Via a link

- Common image formats supported by most web browsers are:
  - .jpg/.jpeg
  - .png
  - .gif
  - .svg

Image Attributes

We can also use some more attributes for the image, such as:

- Align
- Border
- Float

Embedding Media

- We can directly embed the media using audio and video tags
- Only 3 commonly used audio and video formats are allowed to be used
- Web browser does not require any plugin to play those Audio/Video formats

HTML5 Audio Formats

Browser	.mp3	.wav	.ogg
Firefox	✓	✓	✓
Chrome	✓	✓	✓
Opera	✓	✓	✓
Safari	✓	✓	X
Internet Explorer	✓	X	X

HTML5 Video formats

Browser	.mp4	.webm	.ogv
Firefox	✓	✓	✓
Chrome	✓	✓	✓
Opera	✓	✓	✓
Safari	✓	X	X
Internet Explorer	✓	X	X

Common Media Attributes

- Preload
- Autoplay
- Loop
- Mute
- Media Group
- Poster (Only for video tag)
- Width & height (Only for video tag)

HTML Elements

- Block level elements
  - Block level elements are used to create individual blocks in the HTML document
  - Each block level element starts on a new line
  - It also creates a line break at the start and end on its own
  - We can create a block element inside another block element
- Inline elements
  - Inline elements can be added anywhere in the line
  - It won't create any line break
  - They are individual tags

Block Level Elements

- Paragraph Tag
- Heading Tags
- List Tags
- Div Tag
- Form Tag
- Table Tag

Inline Elements

- Bold, Italic, Underline
- Superscript and Subscript
- Emphasize, Big, Small, Strong
- Insert, Delete, Code, Cite
- Span

div tag

- Division or div tag is a block level tag
- With the help of div tag and CSS, we can create:
  - Groups for a set of data presentation and
  - Web layouts
- We can create the same look using Table tag also
- But that is not a good practice for creating blocks or layouts
- Creating a block or layout using div tag is more easy and flexible
- But it requires lot of practice to become an expert in div tag for creating good layouts

span tags

- Span tag can be
- Used to group the inline elements
- Used to set different styles for a particular section
- Inserted anywhere inside any tag

HTML5 Layouts

- In HTML5 we can also create layouts without the div tag
- For each part of the webpage, we have the layout elements

  <header> - Page headers

  <nav> - Navigations or Links

  <section> - Document sections

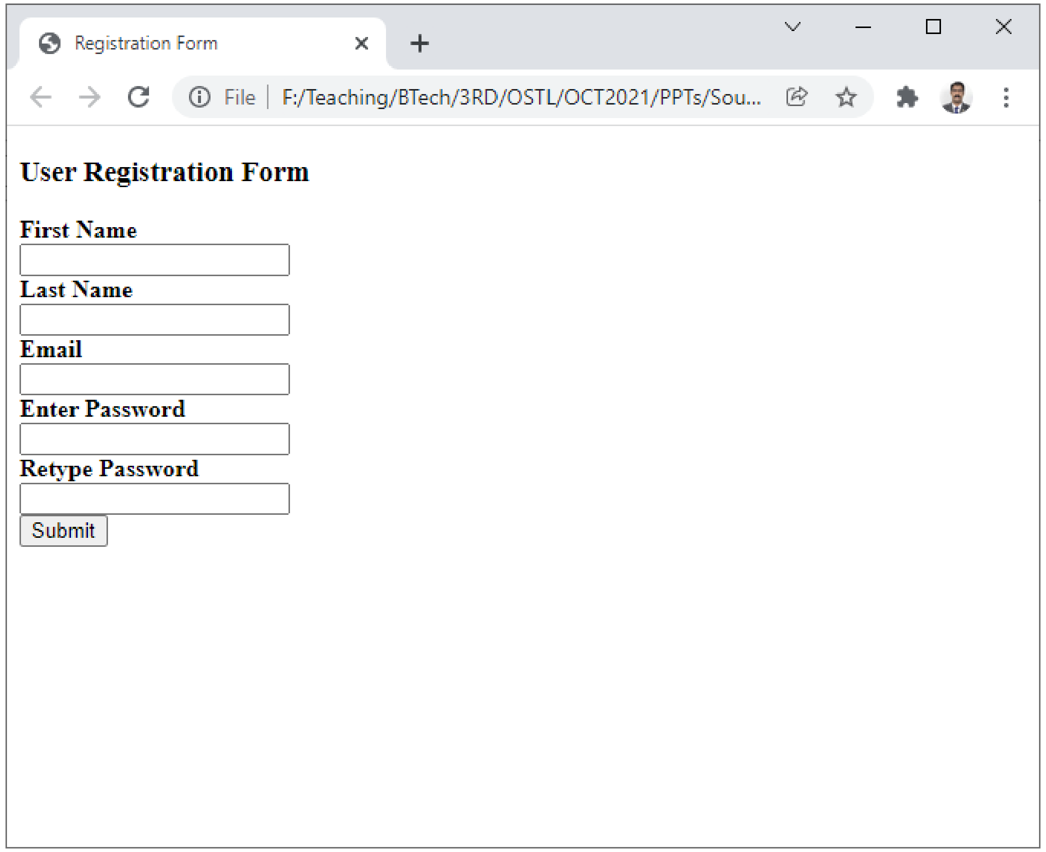
  <article> - Articles

  <aside> - Sidebar contents

  <footer> - Page footer

Web Forms

- Forms are generally used to collect some information from the user
- It can be a
  - Registration Form
  - Login Form
  - Survey
  - Billing Form



User Registration Form

First Name

Last Name

Email

Enter Password

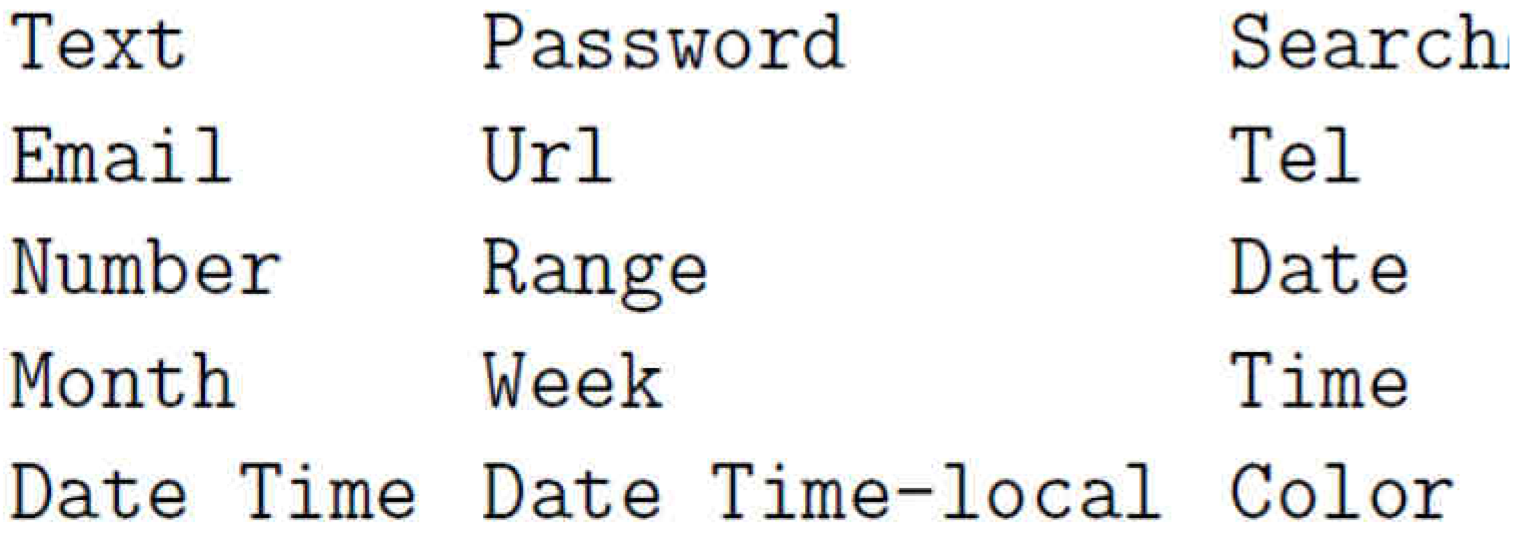
Retype Password

Form Elements

- Label
- Input
- Text Area
- Select
- Datalist
- Output
- Fieldset
- Button

Input Types

- We can use any one of the following input types based on our requirement



Text	Password	Search
Email	Url	Tel
Number	Range	Date
Month	Week	Time
Date Time	Date Time-local	Color

Button Element

- Button element is used to perform an action when the button is clicked

For eg:

- store the input data to the database
- send the form values to another page or script, to perform another action
- display some message on the same page

Date/Time Input type

- Date
- Month
- Week
- Time
- Date Time
- Date Time-local

Form Methods

- Form method will be used when we are working with
- Interactive web pages
- Client-server mechanism

Two methods

- GET
- POST

GET Method

- GET method will pass the values via URL
- Passed values will be seen in the URL

  syntax: <form action="form2.html“ method="get">

- action attribute defines the place to which the values have to be passed

POST Method

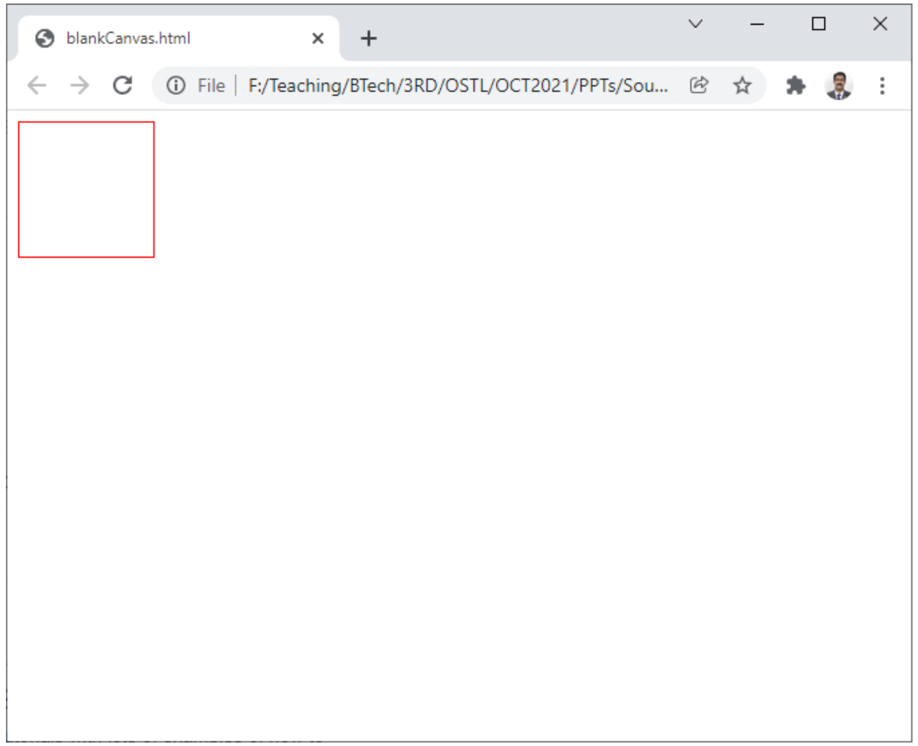
- POST method will pass the values via HTTP request
- Passed values will not be seen in the URL
- It is advisable to use POST method for secure transactions like login and money transactions

  syntax: <form action="form2.html“ method="post">

- action attribute denes the place to which the values have to be passed

HTML Canvas

- HTML5 element <canvas> gives you an easy and powerful way to draw graphics using JavaScript. It can be used to draw graphs, make photo compositions or do simple (and not so simple) animations.
- Here is a simple <canvas> element which has only two specific attributes **width** and **height** plus all the core HTML5 attributes like id, name and class, etc.



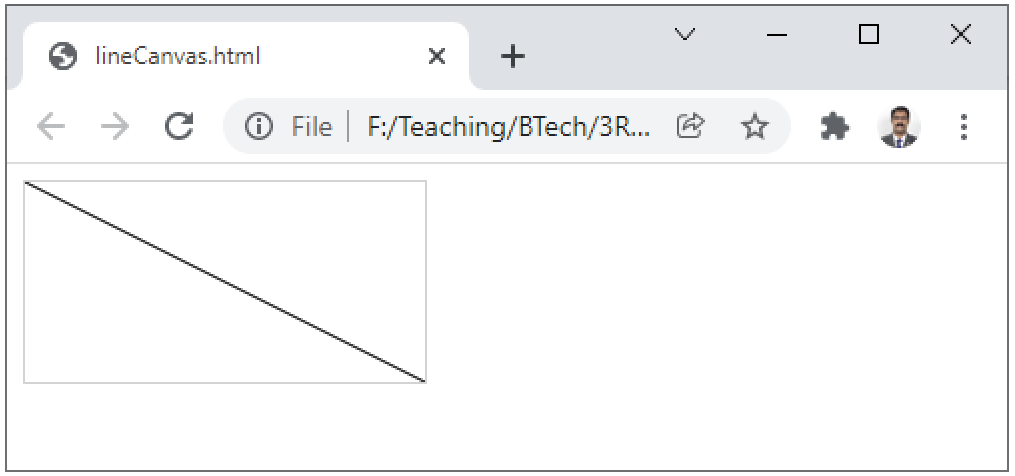
```
<canvas id="mycanvas" width="100" height="100"></canvas>
```

Rendering Context

- The <canvas> is initially blank, and to display something, a script first needs to access the rendering context and draw on it.
- The canvas element has a DOM method called **getContext**, used to obtain the rendering context and its drawing functions. This function takes one parameter, the type of context **2d**.
- After creating the rectangular canvas area, you must add a Script to do the drawing.

Draw a line

```
<!DOCTYPE html>
<html> <body>
<canvas id="myCanvas" width="200" height="100" style="border:1px solid #d3d3d3;">
Your browser does not support the HTML canvas tag.</canvas>
<script>
  var c = document.getElementById("myCanvas");
  var ctx = c.getContext("2d");
  ctx.moveTo(0,0);
  ctx.lineTo(200,100);
  ctx.stroke();
</script>
</body> </html>
```



What to do now …

- https://www.w3schools.com/
- https://www.tutorialspoint.com/html5/
- …
- …
- …
- …

Thank you ☺

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Cascading Style Sheets

Open source Technologies Lab

Topics

1. CSS?
2. **Understanding Style Rules**
3. Three Ways to Insert CSS
4. Examples
5. Advantages & Disadvantages

CSS?

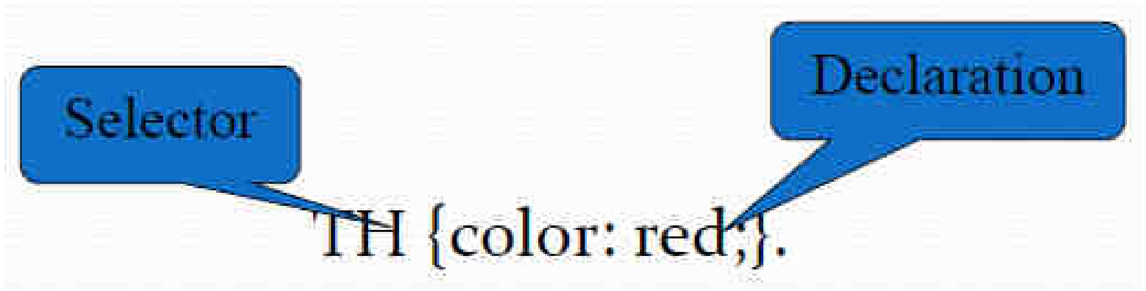
- **CSS** stands for **C**ascading **S**tyle **S**heets
  - Styles define **how to display** HTML elements
- CSS has various levels and profiles. Each level of CSS builds upon the last, typically adding new features and typically denoted as CSS1, CSS2, and CSS3.
- **The first CSS** specification to become an official W3C Recommendation is CSS level 1, published in December 1996
- **CSS level 2** was developed by the W3C and published as a Recommendation in May 1998. A superset of CSS1, CSS2 includes a number of new capabilities like absolute, relative, and fixed positioning of elements and z-index, the concept of media types etc.
- **CSS 3 l**evel 3 is current one. The W3C maintains a CSS3 progress report.

CSS

- CSS Saves a Lot of Work!
- CSS defines HOW HTML elements are to be displayed.
- Styles are normally saved in external .css files. External style sheets enable you to change the appearance and layout of all the pages in a Web site, just by editing one single file!

Understanding Style Rules

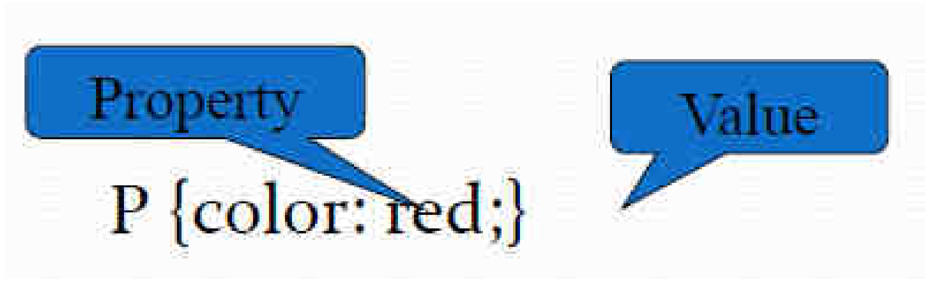
- A Style Rule is composed of two parts: a selector and a declaration.



- The Selector indicates the element to which the rule is applied.
- The Declaration determines the property values of a selector.

Understanding Style Rules

- The Property specifies a characteristic, such as color, font-family, position, and is followed by a colon (:).
- The Value expresses specification of a property, such as red for color, arial for font family, 12 pt for font-size, and is followed by a semicolon (;).



CSS

- **CSS declarations** always ends with a semicolon, and declaration groups are surrounded by curly brackets:

```
p { color:red; text-align:center; }
```

- To make the CSS more readable, you can put one declaration on each line, like this:
- Example

```
p{color:
red;
text-align:center;
}
```

- **CSS Comments**

A CSS comment begins with "/*", and ends with "*/"

```
/*This is a comment*/
p {
text-align:center;
/*This is another comment*/
color:black;
font-family:arial;
}
```

The id and class Selectors

- In addition to setting a style for a HTML element, CSS allows you to specify your own selectors called "id" and "class".

The id Selector

- The id selector is used to specify a style for a single, unique element.
- The id selector uses the id attribute of the HTML element, and is defined with a "#".

The style rule below will be applied to the element with id="para1"

Example

```
#para1
{
text-align:center;
color:red;
}
```

The class Selector

- The class selector is used to specify a style for a group of elements. Unlike the id selector, the class selector is most often used on several elements.
- This allows you to set a particular style for any HTML elements with the same class.
- The class selector uses the HTML class attribute, and is defined with a "."

The class Selector

- In the example below, all HTML elements with class="center" will be center-aligned

Example

```
.center
{
text-align:center;
}
```

In the example below, all p elements with class="center" will be center-aligned

Example

```
p.center { text-align:center; }
```

Three Ways to Insert CSS

- External style sheet
- Internal style sheet
- Inline style

External Style Sheet

- An external style sheet is ideal when the style is applied to many pages. With an external style sheet, you can change the look of an entire Web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section

```
<head>
<link rel="stylesheet" type="text/css" href="mystyle.css" />
</head>
```

- An external style sheet can be written in any text editor. The file should not contain any html tags. Your style sheet should be saved with a .css extension.

Example

```
hr { color:red;}p {margin-left:20px; }
body { background-image:url("images/back40.gif"); }
```

Internal Style Sheet

- An internal style sheet should be used when a single document has a unique style.

```
<head>
<style type="text/css">
    hr {color:red;}
    p {margin-left:20px;}
    body {background-image:url("images/back40.gif");}
</style>
</head>
```

Inline Styles

```
<p style="color:red; margin-left:20px">This is a paragraph.</p>
```

Multiple Style Sheets

- If some properties have been set for the same selector in different style sheets, the values will be inherited from the more specific style sheet.

For example, an external style sheet has these properties for the h3 selector

```
H3
{
color:red;
text-align:left;
font-size:8pt;
}
```

And an internal style sheet has these properties for the h3 selector

```
H3
{
text-align:right;
font-size:20pt;
}
```

If the page with the internal style sheet also links to the external style sheet the properties for h3 will be

```
color:red;
text-align:right;
font-size:20pt;
```

The color is inherited from the external style sheet and the text-alignment and the font-size is replaced by the internal style sheet.

More examples

```
body {background-color:#b0c4de;}
h1 {background-color:#6495ed;}
p {background-color:#e0ffff;}
div {background-color:#b0c4de;}
```

```
body
{
background-image:url('paper.gif');
}
```

```
Body
{
background-
image:url('img_tree.png');
background-repeat:no-repeat;
}
```

```
Body
{
background-
image:url('gradient2.png');
background-repeat:repeat-x;
}
```

More examples

```
ul.a {list-style-type: circle;}

ul.b {list-style-type: square;}
```

```
table, th, td
{
border: 1px solid black;
}
```

More examples

```
<style>
.contentBox
{
        display:block;
        border-width: 1px;
        border-style: solid;
        border-color: 000;
        padding:5px;
        margin-top:5px;
        width:200px;
        height:50px;
        overflow:scroll
}
</style>
<div class="contentBox"> Why do I call them "CSS Scrollbars"?
</div>
```

More examples

```
<html> <head>
<style type="text/css">
p {
background-color:yellow;
}
p.padding
{
padding-top:25px; padding-bottom:25px; padding-right:50px; padding-left:50px;
}
</style>
</head>
<body>
<p>This is a paragraph with no specified padding.</p>
<p class="padding">This is a paragraph with specified paddings.</p>
</body>
</html>
```

Advantages of CSS

- **CSS saves time** When most of us first learn HTML, we get taught to set the font face, size, colour, style etc every time it occurs on a page. This means we find ourselves typing (or copying & pasting) the same thing over and over again. With CSS, you only have to specify these details once for any element. CSS will automatically apply the specified styles whenever that element occurs.
- **Pages load faster** Less code means faster download times.
- **Easy maintenance** To change the style of an element, you only have to make an edit in one place.
- **Superior styles to HTML** CSS has a much wider array of attributes than HTML.

Disadvantages of CSS

- **Browser compatibility** Browsers have varying levels of compliance with Style Sheets. This means that some Style Sheet features are supported and some aren't. To confuse things more, some browser manufacturers decide to come up with their own proprietary tags.

What to do now …

- https://www.w3schools.com/
- …
- …
- …
- …
- …

Thank you ☺

{.js} JavaScript

Open source Technologies Lab

Topics

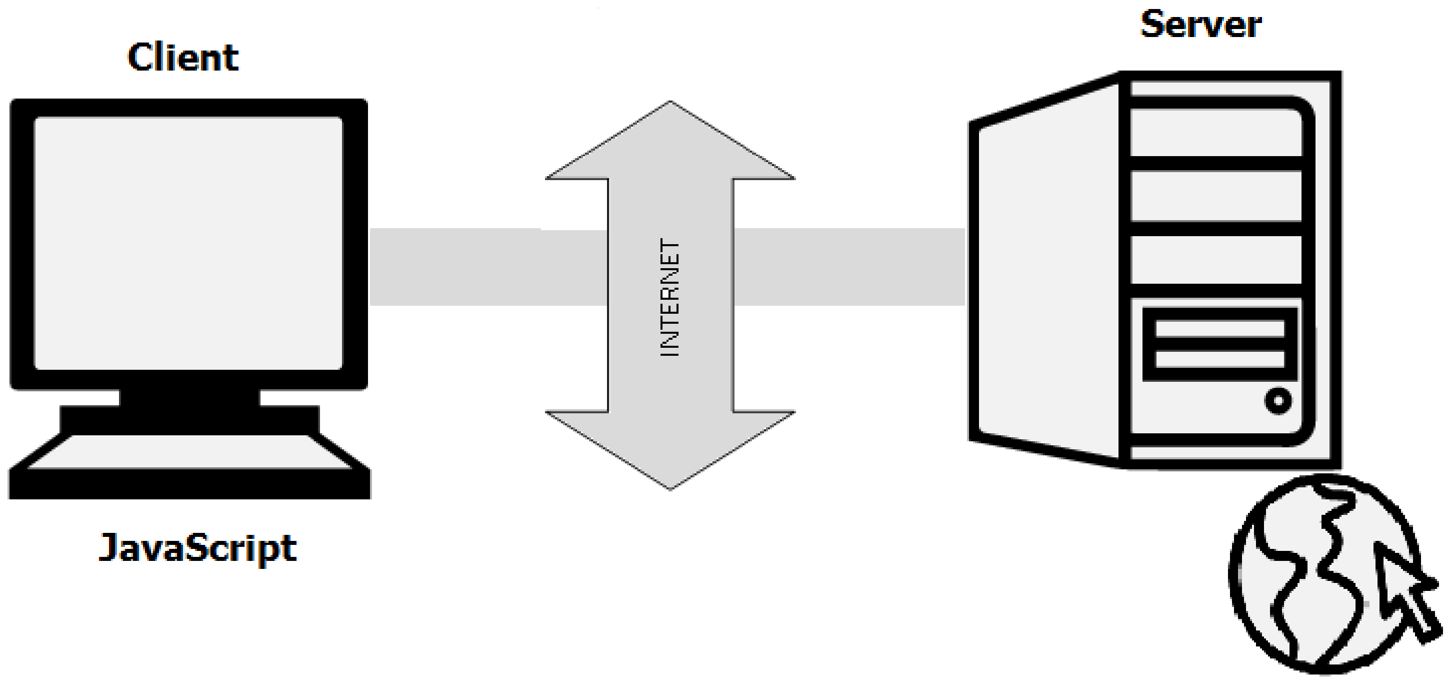
1. What is JavaScript?
2. JavaScript Variables
3. JavaScript Array Methods
4. JavaScript Loops
5. Conditional Statements
6. JavaScript Define & Call Functions
7. JavaScript DOM Tutorial
8. Javascript Examples

What is JavaScript?

- JavaScript is a very powerful **client-side scripting language**. JavaScript is used mainly for enhancing the interaction of a user with the webpage. In other words, you can make your webpage more lively and interactive, with the help of JavaScript. JavaScript is also being used widely in game development and Mobile application development.

New Features

- JavaScript is a **client-side scripting language** developed by Brendan Eich.
- JavaScript can be **run on any operating systems** and almost all web browsers.
- You need a text editor to write JavaScript code and a browser to display your web page.



MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

How to Run JavaScript?

- Being a scripting language, **JavaScript cannot run on its own. In fact, the browser is responsible for running JavaScript code**. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it. The main advantage of JavaScript is that **all modern web browsers support** JavaScript. So, you do not have to worry about whether your site visitor uses Internet Explorer, Google Chrome, Firefox or any other browser. JavaScript will be supported.
- Also, JavaScript **runs on any operating system** including Windows, Linux or Mac.

A Simple JavaScript Program

- You should place all your JavaScript code within **<script> tags** (<script> and </script>) if you are keeping your JavaScript code within the HTML document itself.
- This helps your browser distinguish your JavaScript code from the rest of the code. As there are other client-side scripting languages.
- It is highly recommended that you specify the scripting language you use.
- You have to use the type attribute within the <script> tag and set its value to text/javascript

```
<script type="text/javascript">
```

Hello World Example

```
<html>
<head>
     <title>My First JavaScript code!!!</title>
     <script type="text/javascript">
          alert("Hello World!");
     </script>
</head>
<body>
</body>
</html>
```

Note: type="text/javascript" is not necessary in HTML5.

JavaScript Variable

Declare, Assign a Value

- Variables are used to **store values** (name = “John”) **or expressions** (sum = x + y).

Declare Variables in JavaScript

- Before using a variable, you first need to declare it. You have to use the keyword **var** to declare a variable like this:

```
var name;
```

Assign a Value to the Variable

```
var name = "John";
```
or

```
var name;
name = "John";
```

Naming Variables

- Though you can name the variables as you like, it is a good programming practice to give descriptive and meaningful names to the variables.
- Moreover, variable names should start with a letter and they are case sensitive. Hence the variables studentname and studentName are different because the letter n in a name is different (n and N).

Naming Variables

We will see an example...

JavaScript Array Methods

What is an Array?

- An array is an object that can store a **collection of items**. Arrays become really useful when you need to store large amounts of data of the same type.
- Suppose you want to store details of 500 employees.
- You can access the items in an array by referring to its **indexnumber** and the index of the first element of an array is zero.
- You can create an array in JavaScript as

```
var students = ["John", "Ann", "Kevin"];
```

JavaScript Array Methods

Here, you are initializing your array as and when it is created with values “John”, “Ann” and “Kevin”.The index of “John”, “Ann” and “Kevin” is 0, 1 and 2 respectively. If you want to add more elements to the students array, you can do it like this:

```
students[3] = "Emma";
students[4] = "Rose";
```

You can also create an array using Array constructor like this:

```
var students = new Array("John", "Ann", "Kevin");
```

or

```
var students = new Array();
students[0] = "John";
students[1] = "Ann";
students[2] = "Kevin";
```

JavaScript Array Methods

- The Array object has many properties and methods which help developers to handle arrays easily and efficiently. You can get the value of a property by specifying arrayname.property.
- **length property**
  - ✓ If you want to know the number of elements in an array, you can use the length property.
- **prototype property**
  - ✓ If you want to add new properties and methods, you can use the prototype property.

JavaScript Array Methods

- You can get output of a method by specifying arrayname.method()
- **reverse method**
  - ✓ You can reverse the order of items in an array using a reverse method.
- **sort method**
  - ✓ You can sort the items in an array using sort method.
- **pop method**
  - ✓ You can remove the last item of an array using a pop method.
- **shift method**
  - ✓ You can remove the first item of an array using shift method.
- **push method**
  - ✓ You can add a value as the last item of the array.

JavaScript Array Methods

We will see an example

JavaScript Loops

There are mainly four types of loops in JavaScript.

1. for loop
2. for/in a loop (explore yourself)
3. while loop
4. do…while loop

for loop

Syntax:

```
for(statement1; statement2; statmment3)
{
        lines of code to be executed
}
```

- The statement1 is executed first even before executing the looping code. So, this statement is normally used to assign values to variables that will be used inside the loop.
- The statement2 is the condition to execute the loop.
- The statement3 is executed every time after the looping code is executed.

for loop example

```
<html>
<head>
   <script type="text/javascript">
     var students = new Array("John", "Ann", "Aaron", "Edwin");
      document.write("<b>Using for loops </b><br />");
       for (i=0;i<students.length;i++)
        {
         document.write(students[i] + "<br />");
        }
   </script>
</head>
<body>
</body>
</html>
```

while loop

Syntax:

```
while(condition)
{
        lines of code to be executed
}
```

- The "while loop" is executed as long as the specified condition is true.
- Inside the while loop, you should include the statement that will end the loop at some point of time. Otherwise, your loop will never end and your browser may crash.

while loop example

```
<html>
<head>
       <script type="text/javascript">
         document.write("<b>Using while loops </b><br />");
         var i = 0, j = 1, k;
         document.write("Fibonacci series less than 40<br />");
         while(i<40)
          {
           document.write(i + "<br />");
           k = i+j;
           i = j;
           j = k;
          }
       </script>
</head>
<body>
</body>
</html>
```

do…while loop

Syntax:

```
do
{
        block of code to be executed
} while (condition)
```

- The do…while loop is very similar to while loop.
- The only difference is that in do…while loop, the block of code gets executed once even before checking the condition.

do..while loop example

```
<html>
<head>
      <script type="text/javascript">
            document.write("<b>Using do...while loops </b><br />");
            var i = 2;
            document.write("Even numbers less than 20<br />");
            do
            {
              document.write(i + "<br />");
              i = i + 2;
            }while(i<20)
      </script>
</head>
<body>
</body>
</html>
```

JavaScript Conditional Statements

1. *If* statement
2. *If…else* statement
3. *If…else If…else* statement

if statement

Syntax:

```
if (condition)
{
        lines of code to be executed if condition is true
}
```

- You can use If statement if you want to check only a specific condition.

if statement example

```
<html>
<head>
       <title>IF Statments!!!</title>
       <script type="text/javascript">
              var age = prompt("Please enter your age");
              if(age>=18)
              document.write("You are an adult <br />");
              if(age<18)
              document.write("You are NOT an adult <br />");
       </script>
</head>
<body>
</body>
</html>
```

If…else statement

Syntax:

```
if (condition)
{
        lines of code to be executed if the condition is true
}
else
{
        lines of code to be executed if the condition is false
}
```

- You can use If….else statement if you have to check two conditions and execute a different set of codes.

If...else statement example

```
<html>
<head>
      <title>If...Else Statments!!!</title>
      <script type="text/javascript">
            // Get the current hours
            var hours = new Date().getHours();
            if(hours<12)
            document.write("Good Morning!!!<br />");
            else
            document.write("Good Afternoon!!!<br />");
      </script>
</head> <body> </body> </html>
```

If…else If…else statement

Syntax:

```
if (condition1)
{
        lines of code to be executed if condition1 is true
}
else if(condition2)
{
        lines of code to be executed if condition2 is true
}
else
{
        lines of code to be executed if condition1 is false and condition2 is false
}
```

- You can use If….Else If….Else statement if you want to check more than two conditions.

If…else If…else statement example

```
<html>
<head>
      <script type="text/javascript">
            var one = prompt("Enter the first number");
            var two = prompt("Enter the second number");
            one = parseInt(one);
            two = parseInt(two);
            if (one == two)
                  document.write(one + " is equal to " + two + ".");
            else if (one<two)
                  document.write(one + " is less than " + two + ".");
            else
                  document.write(one + " is greater than " + two + ".");
      </script>
</head>
<body>
</body>
</html>
```

JavaScript Functions

- Functions are very important and useful in any programming language as they make the code reusable A function is a block of code which will be executed only if it is called. If you have a few lines of code that needs to be used several times, you can create a function including the repeating lines of code and then call the function wherever you want.

JavaScript Function

- Use the keyword **function** followed by the name of the function.
- After the function name, open and close parentheses.
- After parenthesis, open and close curly braces.
- Within curly braces, write your lines of code.

```
function functionname()

{

  lines of code to be executed

}
```

JavaScript Function

```
<html>
<head>
<title>Functions!!!</title>
  <script type="text/javascript">
          function myFunction()
           {
            document.write("This is a simple function.<br />");
           }
                      myFunction();
  </script>
</head>
<body>
</body>
</html>
```

Function with Arguments

Syntax:

```
function functionname(arg1, arg2)
{

  lines of code to be executed


}
```

Function with Arguments: Example

```
<html>
<head>
    <script type="text/javascript">
      var count = 0;
       function countVowels(name)
           {
           for (var i=0;i<name.length;i++)
            {
           if(name[i] == "a" || name[i] == "e" || name[i] == "i" || name[i] == "o" || name[i] == "u")
            count = count + 1;
           }
           document.write("Hello " + name + "!!! Your name has " + count + " vowels.");
          }
     var myName = prompt("Please enter your name");
     countVowels(myName);
    </script>
```

JavaScript Return Value

- You can also create JS functions that return values. Inside the function, you need to use the keyword **return** followed by the value to be returned.

Syntax:

```
function functionname(arg1, arg2)
{

  lines of code to be executed

  return val1;
}
```

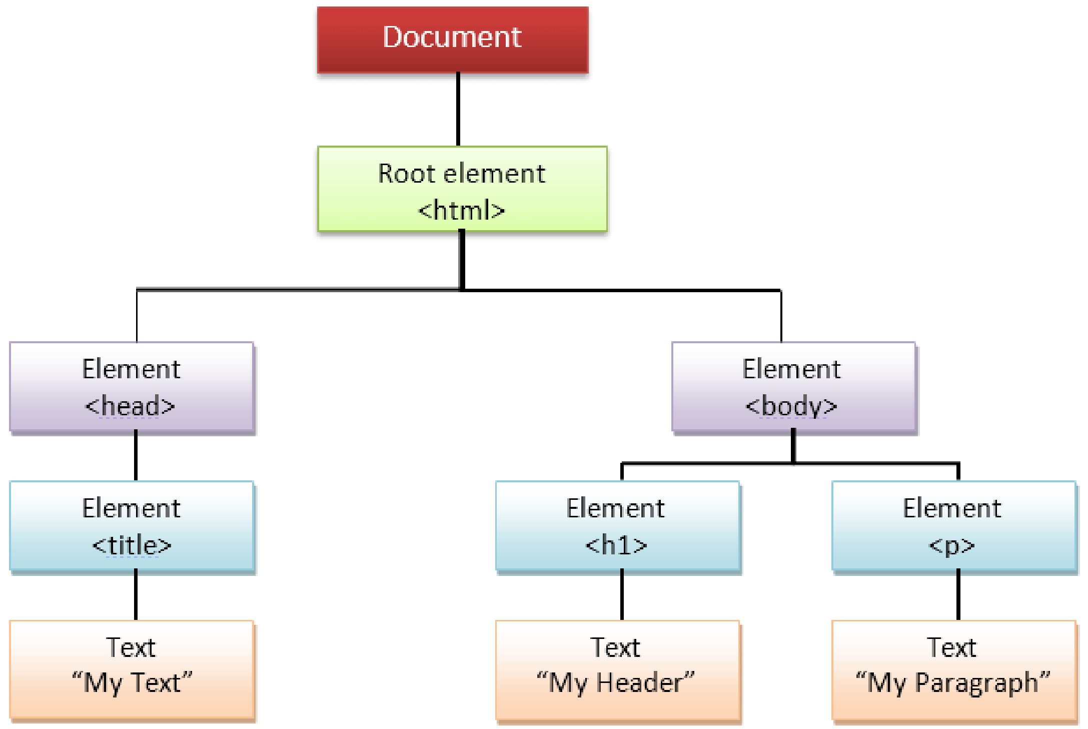
JavaScript Return Value : Example

```
<html>
<head>
  <script type="text/javascript">
   function returnSum(first, second)
    {
      var sum = first + second;
      return sum;
    }
   var firstNo = 78;
   var secondNo = 22;
   document.write(firstNo + " + " + secondNo + " = " + returnSum(firstNo,secondNo));
 </script>
</head>
<body>
</body>
</html>
```

JavaScript DOM

What is DOM in JavaScript?

JavaScript can access all the elements in a webpage making use of Document Object Model (DOM). In fact, the web browser creates a DOM of the webpage when the page is loaded. The DOM model is created as a tree of objects like this:



How to use DOM and Events

- Using DOM, JavaScript can perform multiple tasks.
- It can create new elements and attributes, change the existing elements and attributes and even remove existing elements and attributes.
- JavaScript can also react to existing events and create new events in the page.

getElementById, innerHTML

- getElementById
  - To access elements and attributes whose id is set.
- innerHTML
  - To access the content of an element.

getElementById, innerHTML: Example

```
<html>
<head>
        <title>DOM1!!!</title>
</head>
<body>
 <h1 id="one">Welcome</h1>
 <p>This is the welcome message.</p>
 <h2>Technology</h2>
 <p>This is the technology section.</p>
     <script type="text/javascript">
                var text = document.getElementById("one").innerHTML;
                alert("The first heading is " + text);
     </script>
</body> </html>
```

getElementsByTagName

- getElementsByTagName:
    - To access elements and attributes using tag name. This method will return an array of all the items with the same tag name.

getElementsByTagName

```
<html>
<head>
        <title>DOM2!!!</title>
</head>
<body>
 <h1>Welcome</h1>
 <p>This is the welcome message.</p>
 <h2>Technology</h2>
 <p id="second">This is the technology section.</p>
 <script type="text/javascript">
        var paragraphs = document.getElementsByTagName("p");
  alert("Content in the second paragraph is " + paragraphs[1].innerHTML);
  document.getElementById("second").innerHTML = "The orginal message is changed.";
 </script>
</body>
</html>
```

Event handler

- createElement
  - To create new element
- removeChild
  - Remove an element
- You can add an **event handler** to a particular element like this:

```
document.getElementById(id).onclick=function()
    {
        lines of code to be executed
    }
```

OR

```
document.getElementById(id).addEventListener("click", functionname)
```

Event handler

```
<html>
<head>
        <title>DOM!!!</title>
</head>
<body>
 <input type="button" id="btnClick" value="Click Me!!" />
 <script type="text/javascript">
          function document.getElementById("btnClick").addEventListener("click", clicked);
clicked()
  {
                  alert("You clicked me!!!");
  }
 </script>
</body> </html>
```

JavaScript Multiplication Table

- Create a simple multiplication table asking the user the number of rows and columns he wants.

let us see the code …

POPUP Message using Event

- Display a simple message “Welcome!!!” on your demo webpage and when the user hovers over the message, a popup should be displayed with a message “Welcome to my WebPage!!!”.

JS Forms

Create a sample form program that collects the first name, last name, email, user id, password and confirms password from the user. All the inputs are mandatory and email address entered should be in correct format. Also, the values entered in the password and confirm password textboxes should be the same. After validating using JavaScript, In output display proper error messages in red color just next to the textbox where there is an error.

What to do now …

- https://www.w3schools.com/
- …
- …
- …
- …
- …

Thank you ☺