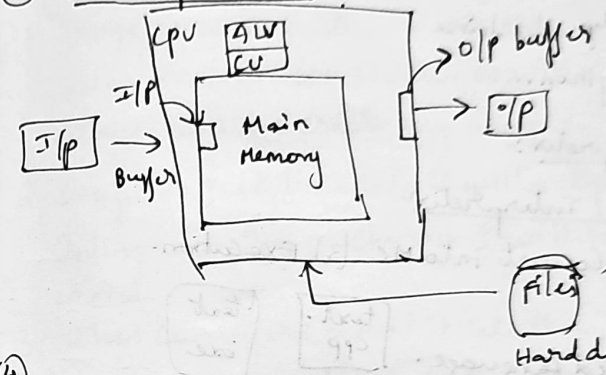
③ How Computer works ?



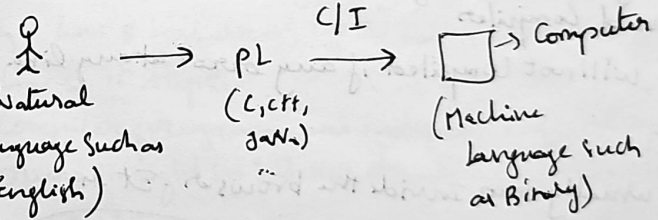
④

→ Binary system.

Electronics works on binary system.

⑤ program :-

procedures need to give to Machine



⑥ Low level Vs High level :-

Low level :- Assembly, Machine

High level :- C, C++, Python....

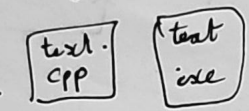
⑦ Compiler Vs Interpreter :-

Function of Compiler / Interpreter :-

(1) Check Errors (2) Convert into MC (3) Execution.

Compiler :-

C++ → Compiler based language.



(1) Compilation (or) translation of source to Machine Code are done only once.

(2) Compiler generates one separate exe Executable file.

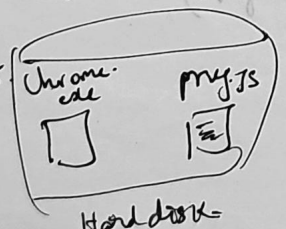
(3) whenever you what do you want to run the program, you don't need Compiler

(4) program will not Compiled if any error at any line.

Interpreter :-

Javascript usually runs inside the browser. It is web-programming.

Chrome is interpreter for Javascript



→ chrome translate and execute line by line.

Differences :-

1. Compiler Convert all line into ML, but make a separate file, browser for JavaScript will act as Interpreter which will not create any exe file.
2. Compiler Just translate, it will not execute but Interpreter that as Chrome it will translate as well as execute.
3. When Compiler Convert it into M.C, as many times we can run exe file but whenever we want to run JS file we should call chrome will run it everytime, it will translate.
4. If any error in code while Compiling, it will not create build file. But in JS, except error lines, other are executed. (Eg :- line 5 has error, up to line 4 execution occurs, after line 5, it stops.)
5. Compiler programs are independent, interpreter programs runs in sight in interpreter.
6. Fast :- Compiler is fast because it is independent but Interpreter runs inside Context of Interpreter so they are slower which ~~are Easy to write~~.

7. Interpreter is easier way to prgm than Compiler because error can be identified at line easy in Interpreter.

(8) Operating System:- - Manage resources.

(9) Programming paradigm / Methodologies:-

→ Monolithic pgm (Basic)
→ procedural/Modular (C-lang)
→ Modular (C-lang)
→ Object-oriented (C++)

(i) Monolithic :- (Basic)

program -  → All in single file, so programmer is only one. → It is first approach when pgm. lgngs. developed.

(ii) procedural/Modular (C-lang) :-

```
fn1()          int main()
{                {
 ---              fn1();
 ---              fn2();
}                 fn2();
fn2()             fn1();
{                }
 ---
 ---
}
```

→ By splitting appliction into smaller, easy to handle
→ More organised.

(iii) Modular :- (C-lang)

```
structure info     main()
{ data1;           {
  data2;           struct info i;
  data3;           fn1(i);
}                  fn2(i);
fn1(Info)          :
{ ---              }
}
fn2(Info)
{ ---
  ---
}
```

→ More organised structure of pgm.
→ Recent pgm.

(iv) OBject-Oriented (C++)

```
Class info      main()
{               {
 data1;          Info i;
 data2;          i.fn1();
 fn1();          i.fn2();
 { ---           ----
   ---           ----
 }               }
 fn2();
 {
   ---
   ---
 }
};
```

→ Here functions are get stored in structure (i.e) class.
→ By creating objects, it can be used for better.

(10) Algorithm :-

→ Step-by-step procedure to solve a prblm.

→ Code (English - Simple) written in any representation called pseudo-code.

Pgm → followed by Machine, so programmer must follow the syntax of C, C++.

(11) Flowchart :-

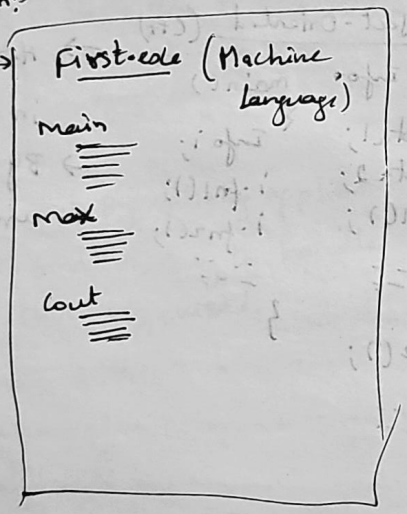
It is used to understand the program flow.

It is not now used but still in academic.

(12) Development & Execution :-

First.cpp

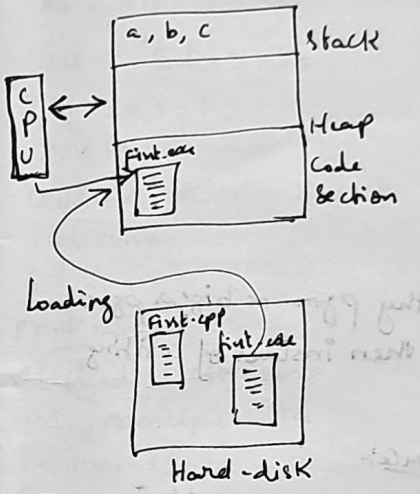
```
#include<iostream.h>
int max (int x, int y)
{
  if(x>y)
   return x;
  else
   return y;
}
int main()
{
 int a=10, b=5, c;
   c = max(a,b);
   cout<< c;
}
```



1. Editing
2. Compiling
3. Linking Library
4. Loading
5. Execution

→ Development stage.

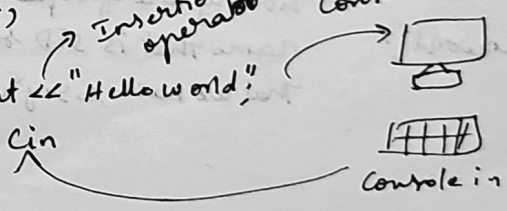
All in one place IDE



→ After loading, operating system make request to cpu to run/execute the code in Code section.

(21) Skeleton of program :-

```
#include <iostream>
int main()
{
  cout << "Hello world!";
}
```



Cout is present in Library iostream. Some Computers iostream.h (or) just iostream

First.cpp

```
#include <iostream>

int main()
{
std::cout<<"Helloworld";
return 0;
}
```

It is perfect pgm.

Now if we are writing very lengthy pgm or bigger pgm we have to use cout many times then instead of writing it like.

Other method:-

```
#Include <iostream>
using namespace std;
int main()
{
cout<<"Helloworld";
return 0;
}
```

namespace:-

All build in things available in this header (Iostream) that is library are grouped under one name that is STD so for using that we have to say "using namespace".

22. How to write a pgm?

add:-

```
#include <iostream>
using namespace std
int main()
{
int c,a,b;
Cout<<"Enter 2 no.s";
Cin>>a>>b;
c=a+b;
Cout<<"Addition is"<<c;
return 0;
}
```

Pseudo Code:-

Algorithm for add:-

```
Begin
  print "Enter 2 no",
Read, a,b
  C← a+b
  print "Addition is" c
End.
```

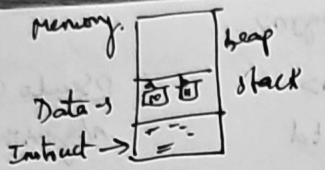
print user name:-

```
#include <iostream>
using namespace std
int main()
{
String name;
Cout<<"May I know your name";
Cin>>name; ⟶ (or) getline(Cin, name);
Cout<<"your name is"<<name;
return 0;
}
```

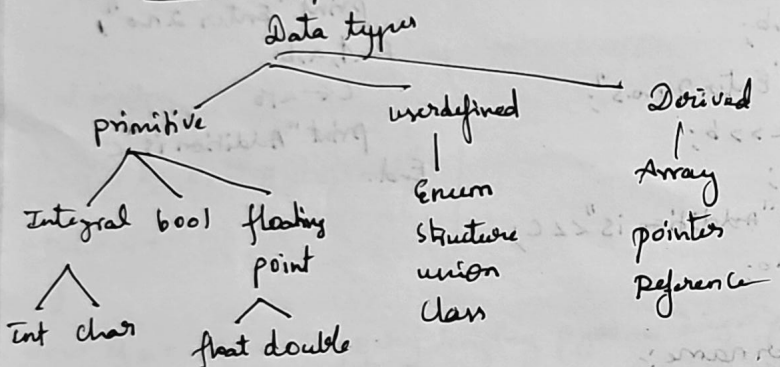
→ for getting all words until press Enter.

24. Data types:-

Char, Integer, float



25. primitive Data types:-

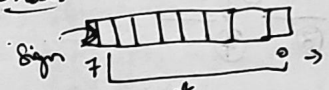


Datatype	size	Range
int	2(or)4	-32768 to 32767
float	4	-3.4×10^{-38} to 3.4×10^{38}
double	8	-1.7×10^{-308} to 1.7×10^{308}
char	1	-128 to 127
bool	undefined	True / false.

Two byte:

$2^{15} \Rightarrow 32768$ (i.e) 0-32767 & -32768 to 32767

char:-



$\Rightarrow 2^7 \Rightarrow 128$ 0-127 & -128 to 127

ASCII

A-65	a-97	0-48
B-66	b-98	1-49
:	:	
Z-90	z-122	9-57

Modifiers: unsigned, long

unsigned int 0-65535

unsigned char 0-255

long int → 4 bytes (If int is 2)
(or)
8 bytes (If int is 4)

long double - 10 bytes

26. Variable :- float a = 10.3f; → need to write for better

27. Arithmetic operation:

int a=3, b=5;

float :- float c = (float)(a/b);

float a=3.5, b=4.1, c;

c = a/b;

But float c = a % b;

×

Mod cannot be applied on float variable.

28. Precedence & Expressions:-

operator	Assumed precedence
( )	3
*, /, %	2
+, -	1

→ left to right

x = a + b * c - d / e

① ② ③ ④

x = (a+b) * (c-d) / e

A = 1/2 bh ⇒ A = b*h/2;

p = 2(l+b) ⇒ p = 2*(l+b);

Sum of n terms $S = \frac{n(n+1)}{2}$ ⇒ s = n*(n+1)/2;

nth term A.P sum t = a+(n-1)d ⇒ t = a+(n-1)*d;

root of quadratic Eqn $\frac{-b \pm \sqrt{b^2-4ac}}{2a}$ ⇒ r = (-b ± sqrt(b*b-4*a*c))/2*a.

Speed $s = \frac{v^2-u^2}{2a}$ ⇒ s = (v*v - u*u)/(2*a)

29. program using Expressions:

$A = \frac{1}{2}bh$

30. Sum of first natural → n*(n+1)/2;

31. finding roots of quadratic Eqn $ax^2+bx+c=0$;

$r = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$; r1 = (-b+sqrt(b*b-4*a*c))/(2*a)

32. Demo - Expressions

33. Compound assignment +=, -=, *=, /= → works with Arithmetic And Bitwise

34. Demo ↗

35. Increment/Decrement operator:

x++ → post incr; ++x → pre incr; x-- → post decrement; --x → pre decrement.

36. Demo ↗

37. overflow :-

char x = 127;

++x; →

x = 0 1 1 1 1 1 1 1 (sign bit) ⇒ 127

+ 1

1 0 0 0 0 0 0 0

Range of char ⇒ -128 to +127

(i) Value ⇒ 128

(ii) along with sign ⇒ -128.

38. Demo ↗ overflow.

39. Bitwise operators &, |, ~, ^, <<, >>

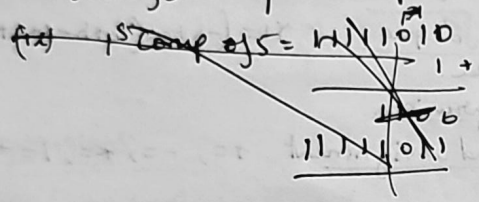
⇓

only works on int, char, not on float

Char x = 5; → 00000101

int y = (int)(~x); → 11111010

↳ To find value of 11111010 is, taking 2's complement for 5;



Finding Value of 1111 1010

1's Comp will be 00000101

2's Comp → 00000101 → 6

Therefore −6.

40. Bitwise demo.

41. Enum & Typedef → month; → Wrong Cannot modify Constant

Enum day {Mon, Tue, wed, Thurs, fri, sat, Sun};

typedef int marks;

typedef int rollno;

int main()

{

marks m1, m2, m3;

rollno r1, r2, r3;

}

for readability of pgm.

Enum → Group of Constants under the name

42. Conditional statements:-

46. Find max of 2 no.s

47. Example

48. Exercise

49. Demo

50. Logical operators

50. Compound Conditional statement:-

51. practice problems

52. Demo

53. Nested if

54.

55.

56. Display Grades

57.

58.

59. Short Circuit

60. Demo

61. Dynamic Declaration

#include<iostream>

using namespace std;

int main()

{

int a=10, b=5;

{

int c=a+b;

if(c>10)

{

}

}

return 0;

}

one method to limit the usage of c variable only for the condition Checking & after that I don't require c.

If I do, →

```
int a=10, b=5;
int c=a+b;
if (c>10)
{
}
return 0;
```

↖ here It will alive throughout

2nd method:- New feature of C++17 extension.

```
int a=10, b=5;
if (int c=a+b; c>10)
{
}
return 0;
```

62. Demo:

63. Switch Case:- → only integral data type is allowed for writing cases. not the float type

64. Demo :-

In else-if, it will check all conditions.

Let say if we enter 7, it should print "Sun". For that it will check all 7 conditions.

To fast this, we are going to use switch case.

67 Arrays :-

In C++, we can write.

```
int A[5] = {1,2,3,4,5};
for(auto x:A)
{
cout<<x<<endl;
}
```

(when we use it, it will automatically identify data type.)

Eg:-

```
for(auto x:A)
{
cout<<++x;
}
```

→ here, it will not modify array contents.

But if we do as below,

Eg:-

```
for (auto &x:A)
{
cout<<++x;
}
```

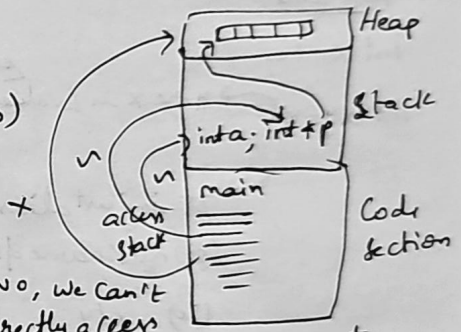
→ now it will update array content. because x holds the address of elements.

108. why pointers?

1) Access heap memory
2) Access files (using file pointers)

.
.
.

Java, C sharp doesn't have pointers so they can't access the devices. only C, C++ able to access devices, so pointers are used in device drivers.



No, we can't directly access the heap, so need to create a pointer in stack, then by using pointer we can access the heap indirectly.

109. Heap memory allocation :-

```
int *p = new int [5] → create array in heap.
delete[]p; → deallocating.
```

Before deallocating, we should not make it Null.

If we do, we can't deallocate the heap memory, because we have missed that address. This is called memory leak.

For making Null as pointer, modern C++ suggested us to use nullptr.

Eg:- p=nullptr;

10. Reference:-

```
int x=10;
int &y=x;
```

Here x is L value.

→ It is just alias (or) nickname of x.

(i.e) x/y → 10

Let say x → 10

int a=x; → a → 10

Above Right value (i.e) x is data not address.

now x=12; → x 12

Above x is L value (i.e) Address

(i) Cout << x is same as Cout << y;

Cout << ++x is same as Cout << ++y;

(ii) But y don't have separate memory. Reference doesn't consume the memory.

(iii) We can't use the same alias to next other variable.

int x=10;

int &y=x;

int b=3;

&y=b.X

16. Introduction to String:

→ ① using character array → in C, C++

② Class string → in C++.

① using character array:-

char a[10]="Hello"; → string literal

char *s="Hello";

char x='A';

char s[]="Hello"; → literal

char s[]={'H','E','L','L','O','\0'}; → without this, it will not considered as string

→ Mostly we use pointers to a string if we want to have string inside heap.

Char *s="Hello"; → literal and literal are created inside Code section

If we want to store string in heap, create Char *s;

If we want to store string in Stack, char s[]={'H','E','\0'};

17. Reading & writing a string:

```
char s[100];
cin >> s;
cout << s << endl;
```

→ steve Jobs → I/P

steve → O/P

```
char s[100];
cin.get(s,100);
cout>>s>>endl;
```

i/p: steve jobs
o/p: steve

```
char s[100];
cin.get(s,100);
cout>>s>>endl;
char s2[100];
cin.get(s2,100);
cout>>s2>>endl;
```

i/p:- Hi buddy
Hi buddy
programe ended

⇒ This will execute but it will not wait for read the i/ps. Because previous i/p contains '\n' : so it '\n' as i/p and stop.

To solve this:-

```
char s[100];
cin.get(s,100);
cout>>s>>endl;
cin.ignore() → (x)
char s2[100];
cin.get(s2,100);
cout>>s2>>endl;
```

① get ≫
Will not take '\n'.
It leave '\n' to Cin.

② getline ≫
It takes '\n' and doesn't leave any i/ps to Cin.

Q:
```
char s4[100];
cin.get(s4,100);
cout<<s4<<endl;
char s2[100];
cin.getline(s2,100);
cout<<s2<<endl;
```

Same Issue:
→ leave '\n' to Cin and '\n' taken by getline and not getting any i/ps.

To solve:
```
char s4[100];
cin.get(s4,100);
cout<<s4<<endl;
cin.ignore() → (x)
char s2[100];
cin.getline(s2,100);
cout<<s2<<endl
```

#18. Length, Concatenate, Copy:-

<string> → header file.

strlen(str);

strcat(destination, source);

strncat(destination, source, n);

strcpy(destination, source);

strncpy(destination, source, n);

119. Substring:

```
char s1[20]="programming"
char s2[10]="gram";
cout << strstr(s1,s2) << endl;
```
⇒ o/p: gramming

```
char s3[10]="m";
cout << strstr(s1,s3) << endl;
```
⇒ o/p: mming.

```
char s4[10]="k";
cout << strstr(s1,s4) << endl;
```
⇒ error (if) Null. Code crash. So need to check null before cout.

```
char s1[20]="programming";
cout << strchr(s1,'g');   → o/p: gramming
cout << strrchr(s1,'g');  → o/p: g
cout << strchr(s1,'r');   → o/p: rramming
```

Compare:-

```
strcmp(s1,s2);
```

120. string to integer, float.

```
char s1[10]="235";
char s2[10]="54.78";
long int x=strtol(s1,Null,10);
float y=Strtof(s2,null);
Cout << x+10 << endl << y-5 << endl;
```
o/p:
245
49.78

Tokeniser:-

```
char s1[20]="x=10;y=20;z=35";
char * token=Strtok(s1,"=;");
While (token!=NULL)
{ cout<< token<<endl;
  token= Strtok(NULL,"=;");
}
```
o/p:
x
10
y
20
z
35

Eg: if statement is

```
char * token=Strtok(s1,"=?");
```
o/p:
x
10;y
20;z
35

121. class string:

string str; → object

Cin>>str; → Now it will read lot of characters. But taken only one word.

122. Basics of class string

s.length();

s.capacity();

s.size();

s.resize(30);

s.max_size();

s.clear()

s.Empty

string s="Hello";

o/p: s.length() → 5;

187. string class Iterator:-

```
string str="today";
string::iterator it;
for(it=str.begin(); it!=str.end(); it++)
{
    *it=*it-32;
}
cout<<str;
cout<<endl;
```

o/p:

TODAY

188. Length of string:

① using:-

```
string s="WELCOME";
int i=0, count=0;
while(s[i]!='\0')
{
    count++;
    i++;
}
cout<<s<<endl;
```

o/p: 7

②

```
string str="WELCOME";
string::iterator itr;
for(itr=str.begin(); itr!=str.end(); itr++)
{
    cout++;
}
cout<<count<<endl;
```

131. checking palindrome

```
string str="madam";
string rstr="";
int len=(int) str.length();
rstr.resize(len);
```

```
for(int i=0;j=len-1; i< len; i++;j--)
{
  rev[i]=str[j];
}
rev[len]="\0";
Cout<< rev<<endl;
if (str.Compare(rstr)==0)
Cout<<"palindrome"<<endl;
else
Cout<<"not a palindrome"<<endl;
```

i/p:- madam

o/p:- palindrome

132. Find username in email address:-

```
String str="John123@gmail.com";
String username="";
int i=(int)str.find('@');  ⟶ It will ingnore @
String username=str.Substr(0,i);
Cout<<"username is "<< username<<endl;
return 0;
```

o/p:- John123

135. Function overloading:-

In C, we can't use same function name.

In C++, we can use.

```
int add(int x,int y)
{
  return x+y;
}
int add(int x,int y,int z)
{
  return x+y+z;
}
float add(float x, float y)
{
  return x+y;
}
int main()
{
int a=10,b=20,c,d;
  c=add(10,20);
  d=add(10,20,c);
float x=1.3,y=4.5,z;
  z=add(1.3,4.5);
}
```

Here add is the same fn. name. But Compiler C++, differentiate them by no. of parameters and different data types.

Note return type is not considered in function overloading

Valid & Invalid :-

int max(int, int) ✓

float max(float, float) ✓

int max(int, int, int) ✓

float max(int, int) ✗ → Because return type is not considered in fun. overloading.

136. Function template :-

```
int main()
{
int c = max(10,5);
float d = max(10.5f, 6.7f);
}

template <class T>
T max(T x, T y)
{
 if (x > y)
  return x;
 else
  return y;
}
```

Instead of ⇐

```
int max(int x, int y)
{
 if (x > y)
  return x;
 else
  return y;
}
float max(float x, float y)
{
 if (x > y)
  return x;
 else
  return y;
}
```

use case :- Combine the same functions with having same function name but with different data types.

→ Both arguments that are passing to function should be same for template ✓ ✗

Eg:

```
Cout << max(2.3f, 4.3f);  ✓
Cout << max(2.3f, 4.3);   ✗
          ↓       ↓
        float   double
```

139. Default Argument :- → Benefit: We can define overloaded fns. by writing a single fn.

```
int add(int x, int y, int z = 0)
{
 return x+y+z;
}
int main()
{
int a = add(2,5);
int b = add(2,5,8);
int c = add(2,5,0);
```

works with last argument

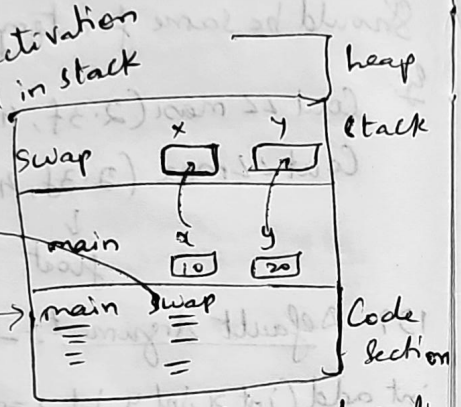
Instead of this ⇐

```
int add(int x, int y)
{
 return x+y;
}
int add(int x, int y, int z)
{
 return x+y+z;
}
```

Q) I call, what is use of this, because it will same as that, why we can use default arg.

14. function call → ① Call by Value
② Call by address
③ Call by reference

```
int swap(int*x, int*y)
{
  int temp
  temp=*a;
  *a=*b;
  *b=temp;
}
int main()
{
int x=10, y=20;
swap(x,y);
}
```

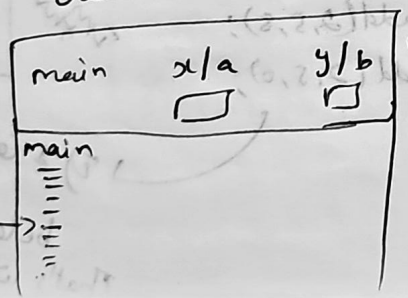


Separate Machine Codes were generated in code section

Let's see what happens if we call fn by reference

```
int swap(int &a, int &b)
{ int temp=a;
  a=b;
  b=temp;
}
int main()
{
int x=10, y=20;
swap(x,y);
}
```

reference in C++
There is no activation record created.



More for swap fn, Code is not generated separately. Here Code is placed in the main function wherever fun Call is there. So here it Can be Called as "Inline functions".

So → *Don't use loops & Complex logic in function which is called by reference, bCoz sometimes Compiler Can't replace and feel Code size big.

14.1 Return by address:

```
int* fn(void)
{
  int *p=new int[5];
  for(int i=0; i<5; i++)
    p[i]=5*i;
  return p;
}
int main()
{
  int *ptr=fn();
  :
}
```

Return by reference

```
int & fn(int &x)
{
  return x;
}
int main()
{
  int a=10;
  fun(a)=25;
  Cout<<a<<endl;
}
```

a/x → 10

→ return reference (i.e) address of a

(i.e) fun(a)=25 is equal to a=25;

o/p:- 25

151. Local vs Global :-

```
int x=10; → Global
int main()
{
  int x=20; → Local
    int x=30; → Local
    Cout<<x<<endl; → o/p:- 30
  }
  Cout<<x<<endl;   20
  Cout<<::x<<endl;  10
}
```

→ Scope resolution operator.

In order to access the global variable but which is already defined as local in a function, we can use Scope resolution operator to use the global variable in the same function.

152. Static variable :-

only difference b/n Static & global variable is that Static variable is only accessed inside the function where it is declared.

157. Introduction to oops :-

Modular programming

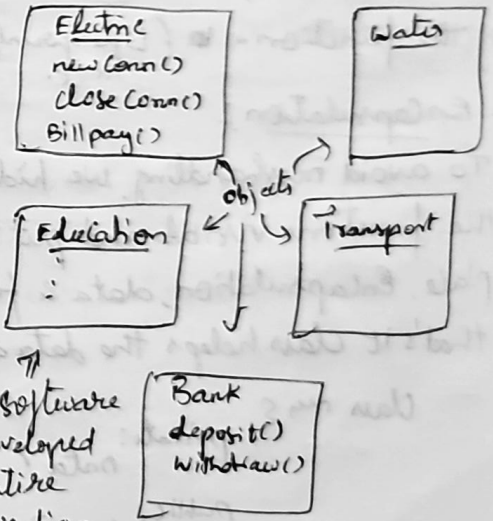
Bank

openACC();
Withdraw();
deposit();

↑ only specific task Software is developed

object-oriented pgming

Govt



Modular :- Collection of functions (very closer look)

OOPS: Collection of objects which defines fns and data related to them (little far away look)

158. principles of object-Orientation :-

Two elements needed for software dev:

① Data ② functions.

1. Abstraction :- class my { Data, functions }

only user ↓ Knows the name of functions and use it, programmer, he doesn't need to know how implementation of the function. (Like printf in C.)

2. Encapsulation :-

To avoid mishandling, we hide the data and make the functions visible and put things together at one place, Encapsulation, data & funcs together, so that's it Class helps the data & fns together.

class my { private: data → Data hiding.

public: fns }

3. Inheritance - Eg: Tdy If we write a class, then tomorrow we want another class in which we want all features of class 1 plus some additional features, so we should be able to inherit, borrow all features from class 1.

Eg: BMW Car version 1 (Existing) ⟶ BMW Car version 2 (version 1's feature + addition feature)

Not fully redesign from scratch

4. polymorphism: People who are using things need not re-learn them just they've learned something in upper heirarchy, (i.e) upper class in heierarchy then they can manage all this.

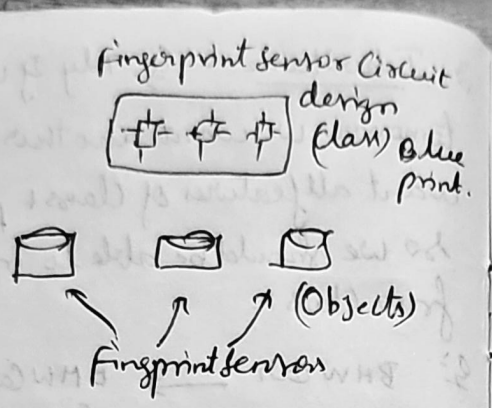
Eg: If we know how to use browser do we need to learn how to use chrome (or) how to use Internet Explorer? - NO, once we know how to use one of the browser, we can access Internet using any browser.

polymorphism → has many forms.

159. class vs object :-

class Human → you
→ me

class Car → BMW
→ Toyato



Objects are created based on class

Data is property, fn is the behavior.

Eg:-

```
class Rectangle
{
  float length;
  float breadth;

  float area();
  float perimeter();
  float diagonal();
};
main():-
  {
    Rectangle r1, r2, r3;
  }
```

objects

class → user defined data-type which has data members & member functions

object → Instance of class. When class is defined, no memory is allocated but when it is instantiated (object is created) memory is allocated.

160. Writing a class:-

```
class Rectangle
{
public:
    int length, breadth;
    int area
    {
      return (len * br);
    }
  int perimeter
  {
    return 2*(length+bread);
  }
};
Void main()
{
Rectangle r1, r2;
r1.length=10;
r1.breadth=5;
r2.len=5; r2.br=10;
cout<< r1.area();
cout<<(r2.area();
}
```

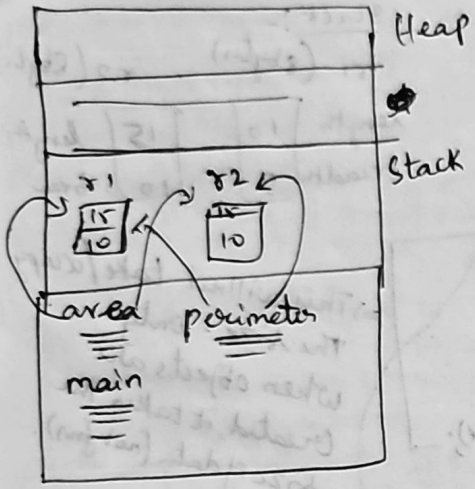
we can't allow if we didn't mention as public. By default, it is private.

Stack :-

r1 (8 bytes) r2 (8 bytes)

	r1	r2	
length	10	15	length
breadth	5	10	brea

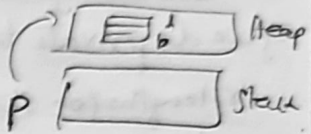
=> This will not take/occupy the size. only when objects are created, it takes the size of data (not fun).



162- pointer to object:-

```
Void main()
{
  Rectangle R;        → stack
  Rectangle *p;       → stack
  p=&R;
  p->length=10;
  p->breadth=15;
  Cout << p->area();
}
```

pointer to object in heap:-




```
void main()
{
  Rectangle *p = new Rectangle;
    p->length=10;
    p->breadth=15;
  Cout << p->area();
}
```

164. Philosophy behind Data Hiding:-

If we made data (length, breadth) as public, we will see what happens.

```
int main()
{
  Rectangle r;
   r.length=5;
   r.breadth= -10;   → user done mistake (giving negative)
  Cout << r.area();  → This give negative area.
}
```

Now, user blames the class that gives the negative area. Here data members are mishandled, so they blame class.

That's why we should give access only to functions not the data members.

165. Data hiding in C++ (Accessors & Mutators)

If we do private to data,

```
r.length=10; X
r.breadth=20; X
Cout<< r.length; X
Cout<< r.area(); ✓ → But this give garbage value. Crt. bloz it won't allow user to Initialize.
```

Then how to use?

Yes, we need to create functions to set length, breadth.

```
class Rectangle
{
 private: int length, int breadth;
 public:
      void setLength(int l)
      { if(l>0)
          length=l;
        else
          length=0;
      }
      void setBreadth(int b)
      { If(b>0)
          breadth=b;
        else
          breadth=0;
      }
      int getLength()
      { return length;
      }
      int getBreadth()
      { return Breadth;
      } → also assume having area, perimeter fns.
};
```

So,

```
Rectangle r1;
r1.setLength(10);
```

Here Accessors → Getxxx();
Mutators → Setxxxx();

167. Philosophy behind Constructors:-

Whatever we have done regarding, philosophically it is wrong.

Let's assume,

When we purchase a car, it has own colour when we ordered. We buy it and after buying we don't colour it. ort? When we purchase rectangular box, after we purchase did we set length & breadth?

→ No. ~~It was~~

- So we don't call setlength, breadth...;
-> We should have a function which should be automatically called when object is created / Constructed that takes parameters and assign those values.

168. Constructor:-

1. Default Constructor
2. Non-parameterized " "
3. parameterized " "
4. Copy Constructor

] user defined Constructor.

Example for 2:-

- go & ask pizza -> They will give most popular/having instock. -> This is non-parameterized Con.

Example for 3:-

-> go & ask cheeze double side pizza -> parameterized

Example for 4:-

-> Go and show the hand to menu and ask the same to shop. They will give the same that you shown from menu -> This is Copy Constructor.

```
Rectangle r;  ->
Rectangle r();  ~>
Rectangle r(10,5);

Rectangle r2(r);
   ↓
Copy
Constructor
```

```
class Rectangle
{
private:
   int length, breadth;
public:
   Rectangle()
   {
      length=0;
      breadth=0;
   }
   Rectangle(int l, int b)
   {
      setlength(l);
      setbreadth(b);
   }
   Rectangle (Rectangle &rect)
   {
      length = Rect.length;
      breadth = Rect.breadth;
   }
};
```

We can also modify

```
class Rectangle
{
 private:
    int length;
    int breadth;
 public:
    Rectangle()
    {
      length=0;
      breadth=0;
    }
  Rectangle(int l=0; int b=0)
  {
    Setlength(l);
    Setbreadth(b);
  }
  Rectangle(Rectangle &rect)
  {
    length=rect.length;
    breadth=rect.breadth;
  }

}
```

} X we don't need. Instead of it,

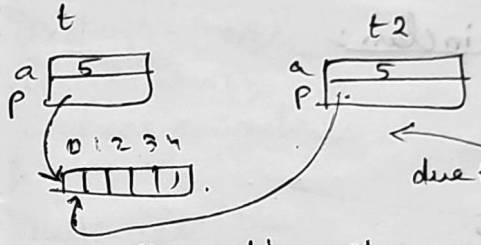
169. Deep Copy Constructor:

```
int main()
{
Test t(5);
Test t2(t);

}
```

```
class Test
{ int a;
  int *p;
  Test(int x)
  {
    a=x;
    p=new int[a];
  }
  Test(Test &t)
  {
    a=t.a;
    p=t.p;
  }
```

So in memory



Here is the problem. When we create object, here they don't create own array, (4) when we do direct memory allocation, we should careful otherwise it points the same as original.

To Solve:-

```
Test(Test &t)
{
  a=t.a;
  p=new int[a];
}
```

→ Now create separate array.

Before Calling Test t2(t) in Code, assume array is already filled with data.

After calling Test t2(t), creates object t2 but there will be no same data of array in t. We should Copy manually.

* Constructor doesn't have return type (X)(X)

171. All types of functions in class :-

```
class Rectangle
{
  private:
      int length, breadth;
  public:
      Rectangle();
      Rectangle(int l, int b);          ] -> Constructors
      Rectangle(Rectangle &r);
      void setLength(int l);            ] -> Mutators
      Void setBreadth(int b);
      int getLength();                  ] -> Accessors
      int getBreadth();
      int area();                       ] -> facilitators
      int perimeter();
      bool isSquare();                  ---> Inquiry
```

~Rectangle(); -> Destructor

}

to

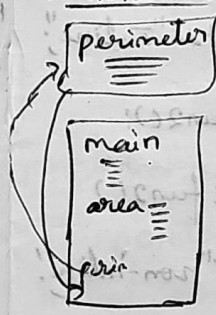
function prototypes are mentioned in there not the definition. functions are defined outside using the Scope resolution operator.

172. Scope resolution operator :-

```
void main()
{
  Rectangle r(10,5);
  Cout<<r.area();
  Cout<<r.perimeter();
}
```

```
class Rectangle{
  private:
      int length, breadth;
  public:
  int area()
  { return (length*breadth);
  }
  int perimeter();
};
int Rectangle::perimeter()
{
  return 2*(length+breadth);
}
```

Machine Code:

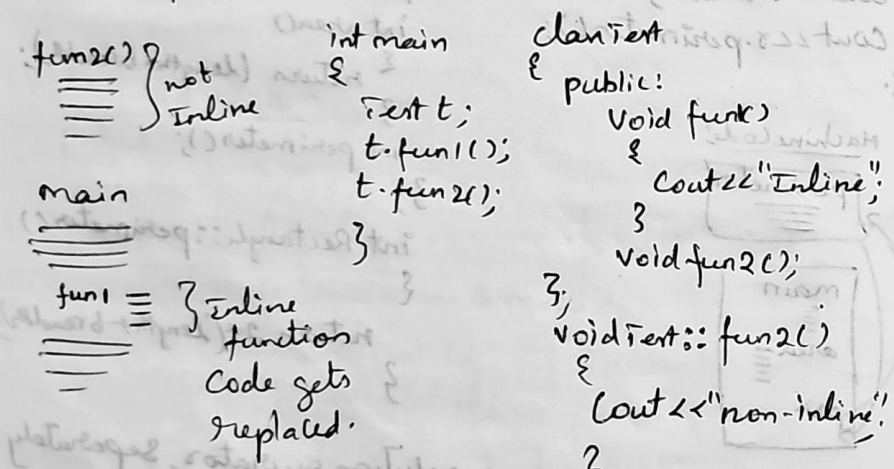


Here when we use scope resolution operator, separately Code is generated for perimeter function where area Code is only replaced at function call only inside the part of main function.

If we don't use Scope resolution and instead of it, if we define all functions inside the class, Separate Code will not be generated. Everything will be replaced when fun. Call happens at main(). So simple at the time area & perimeter are Called Inline functions.

If we use Scope res. op; they are not inline fns.

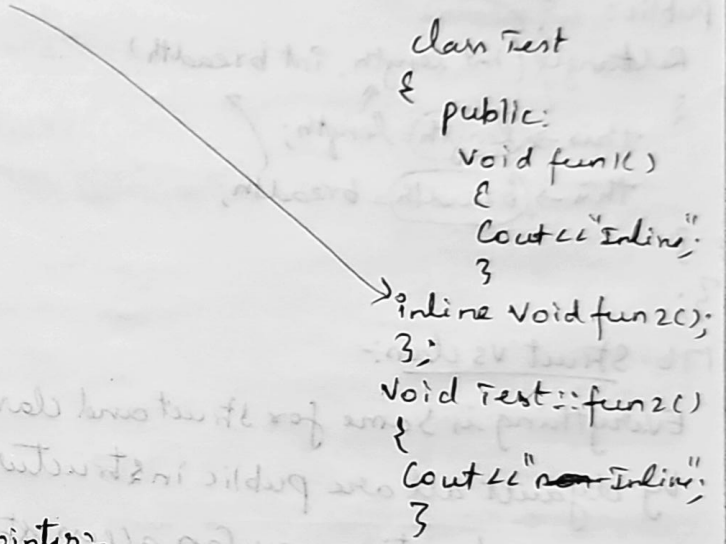
174. Inline functions:-



When we want Inline, we should define function inside the class.

If we don't want it as inline, we can define outside the class using Scope resolution operator.

Other method!



175. This pointer:-

This pointer is used for removing the ambiguity in between the parameters of function with the data member of a class.

```
class Rectangle
{
private:
   int length;
   int breadth;
Public:
  Rectangle (int length, int breadth)
  {
    this -> length = length;
    this -> breadth = breadth;
  }
};
```

176. Struct vs class:-

Everything is same for struct and class except by default all are public in structure.

(ie) In main function you can access structure members like s.x=10;

But in class by default all are private. we can't access in main function.

178. operator overloading:

```
main()
{
   Complex c1(3,7);
   Complex c2(5,4);
   Complex c3;
   c3 = c1.add(c2);
      (or)
   c3 = c2.add(c1);
```

```
class Complex
{
  private:
    int real, img;
  Public:
  Complex(int r=0, int i=0)
  {
    real = r;
    img = i;
  }
  Complex add(complex x)
  {
    Complex temp;
    temp.real = real + x.real;
    temp.img = img + x.img;
    return temp;
  }
```

(or)

c3 = c1.operator+(c2); X

Instead of it,

c3 = c1 + c2;

```
complex operator+(complex x)
{
  .
  .
  .
}
```

This is operator overloading.

179. Friend function:-

C3 = C1 + C2;

Here function is defined without Scope resolution operator.

(i) advantage of:

- This friend function. (has access to private protected data members even outside the class)

```
class Complex
{
    int real;
    int img;
  public:
friend Complex operator+(Complex c1, Complex c2)
};
Complex operator+(Complex c1, Complex c2)
{
  Complex t;
  t.real = c1.real + c2.real;
  t.img = c1.img + c2.img;
  return t;
}
```

182. Insertion operator overloading:-

Eg: cout << C; ← This is not possible. Because cout doesn't know Complex number to print. We should overload;

cout << C

cout → ostream, C → Complex

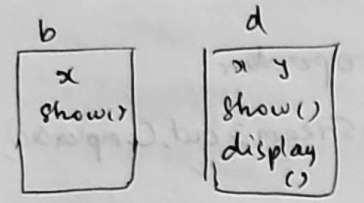
Same as previous class,

```
{
friend ostream& operator<<(ostream &out, Complex &c);
};
ostream& operator<<(ostream &out, Complex &c)
{
    out << c.real << "+i" << c.img << endl;
    return out;
}
int main()
{
  Complex c(10,5);
  cout << c;  → This is also same as below.
  operator<<(cout, c);
}
```

If we use this, we have to define as below.

```
friend void operator<<(ostream &out, Complex &c)
};
void operator<<(ostream &out, Complex &c)
{
  out << c.real << "+i" << c.img << endl;
}
```

185. Inheritance:-




```
int main()
{
  Base b;
  b.x=5;
  b.show(); → 5
  Derived d;
  d.x=10;
  d.show(); → 10
  d.y=15;
  d.display(); → 10 15
  return 0;
}
```

```
class Base
{
  public:
    int x;
  void show()
    {
      cout<<x;
  }; }
class Derived : public Base
{
    public:
      int y;
      void display()
      {
      cout<<x<<" "<<y;
      }
};
```

186. Constructors in inheritance

```
class Base
{
public:
Base()
{cout<<" Default of Base"<<endl;
}
```

```
int main()
{
Derived d;
  ⇓
o/p: Default of Base
     Default of Derived
Derived d(5);
  ⇓
o/p:-
   Default of Base
param Default of Derived 5
Derived d(5,20);
  ⇓
Param of Base 5
param of Derived 20.
```

```
  Base (int x)
  {
  cout<<"param of
  Base"<<x<<endl;
  }
};
class Derived : public Base
{
public:
 Derived()
{
 cout<<"Default of Derived";
}
 Derived(int a)
{
cout<<"param of Derived"<<a;
}
Derived(int x, int a) : Base(x)
{
cout<<"param of Derived"<<a;
}
};
```

191. isA Vs hasA :-

```
class Table
{
    Rectangle top;
    int legs;
    ≡
};
```

```
class Rectangle
{
    ≡
};
```

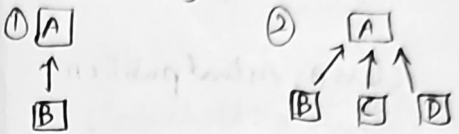
```
class Cuboid : public Rectangle
{
    ≡
};
```

Here Cuboid is also a rectangle.
So Cuboid "is A" relationship with rectangle.
Here Table is having object of Rectangle.
So Table "has A" relationship with rectangle.

192. Access specifiers :-

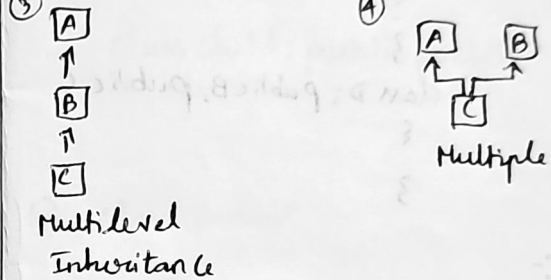
	private	protected	public
Inside the Class	✓	✓	✓
Inside the derived Class	X	✓	✓
On object	X	X	✓

19. Types of Inheritance:-



Simple / Single

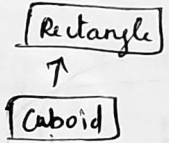
Hierarchical (It is generalization approach)



Multilevel Inheritance

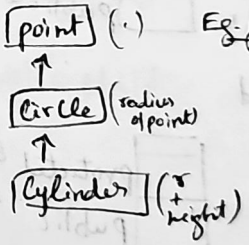
Multiple Inheritance

Eg for Single:-

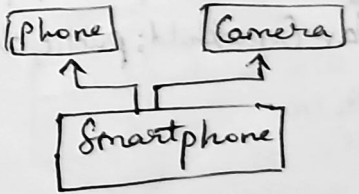


Eg. for hierarchical

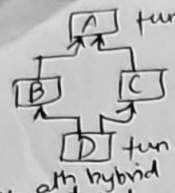
Eg. for Multilevel:-



Eg. for Multiple



⑤



4th hybrid

Multipath Inheritance

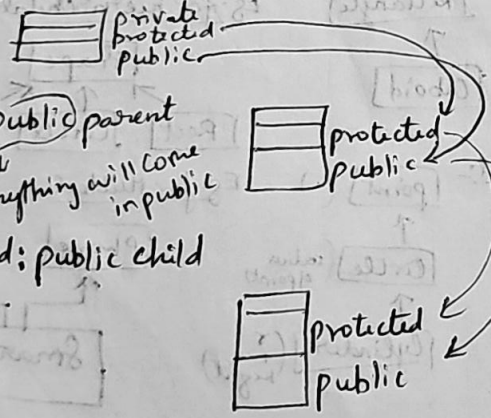
Let's say D has to call fun() in A, how it call whether it via 'B' or 'C'? There is ambiguity.

To solve that, we should use virtual.

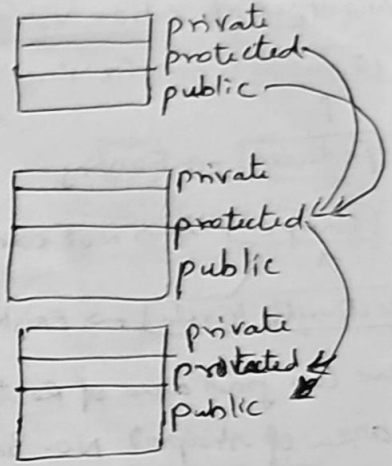
```
class A
{...
}
class B: virtual public A
{...
}
class C: virtual public A
{...
}
class D: public B, public C
{
}
```

195. Ways of Inheritance :-

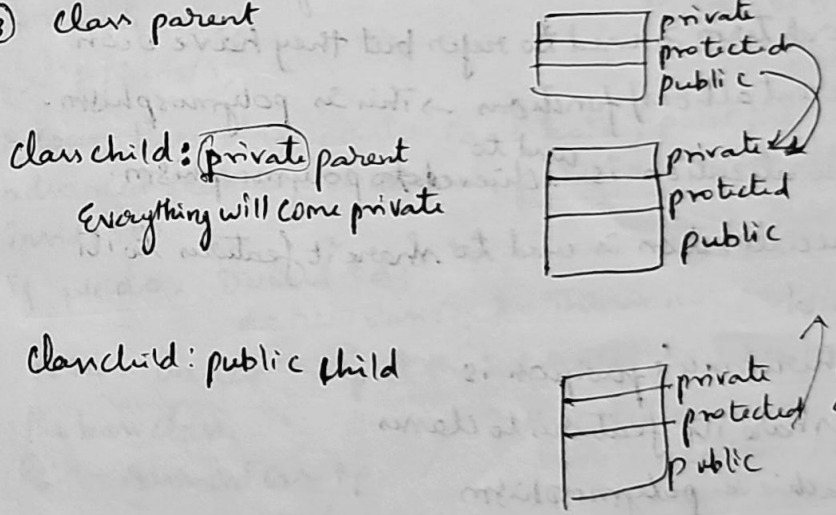
① class parent



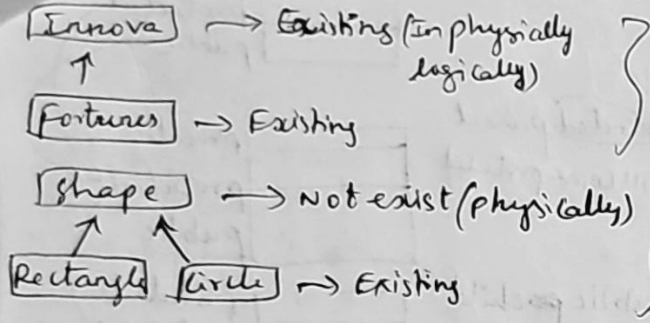
② class parent



③ class parent



197. Specialization VS Generalization :-



We can find area of Rectangle, Circle. Can we find area of shape? No. Because it is general term.

General term is used to refer but they have own different actions/functions → This is polymorphism.

① Generalization is used to achieve polymorphism.

② Specialization is used to share it features to its child

So Inheritance's purpose is

① To share its features to classes

② To achieve polymorphism

199. Base class pointer derived class object :-

```
int main()
{
  Base *p;
  p = new derived();
  p->fun1();
  p->fun2();
  p->fun3();
  p->fun4();
  p->fun5();
```

```
class Base
{
 public:
  void fun1();
  void fun2();
  void fun3();
};
class Derived : public Base
{
 public:
  void fun4();
  void fun5();
};
```

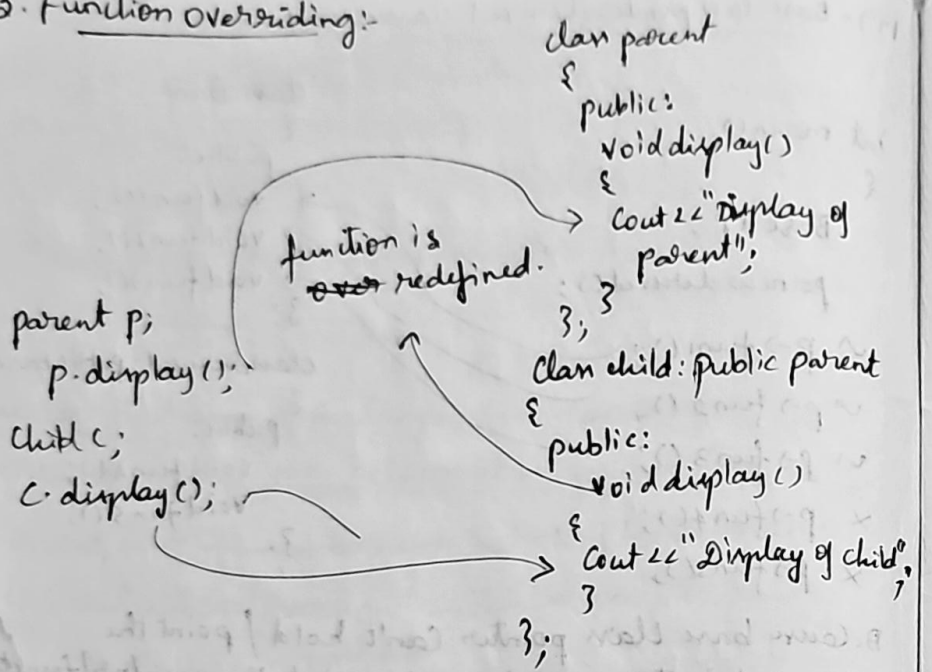
Because base class pointer can't hold / point the advanced features. It can only point the member/functs inside it.

If we do, Derived *d;
d = new Base(); X This is not possible.

Because we can't expect the advanced features inside the base class.

Eg:- AdvancedCars *p; → Here if we assume the basic car contains "AC", "Camera for Car parking" - No. It doesn't - No, It doesn't.
p = new BasicCar();

203. Function Overriding:-



204. Virtual Class:-

```
int main()
{
  Base *p=new Derived();
  p.fun();
```

```
class Base
{
public:
void display()
{
cout<<"fun of Base";
};
```

```
class Derived: public Base
{
public: display.
Void fun()
{
cout<<"fun of derived";
}
};
```

It doesn't Call fun() in derived class.

Note :- very Careful

If they are having different funcn lets say fun1 in base class, fun2 in derived. If we create base pointer p which holds derived class object. p.fun1 can be called. But we can't call p.fun2. Bcoz, fun2 is advanced feature in Derived class so base pointer can't access.

Let's go for an example, I am creating base Car pointer. It points the advance Car. Now I get into the Car. I am thinking it's basic Car. Now I will drive. How it will run? It will run how I am thinking, how I am Calling or it will run what it is? will it run like advanced Car? will it run advance Car mechanism (or) base Car mechanism that I am believing. It will run like an advanced Car. (Below, this is Correct logic)

When you have same function in base class as well as in derived class whose function must be called?

It actually based on object not on pointer

It is just like if you saw a donkey and say it's is horse. you are thinking it a horse, but it's is donkey It will not run like a horse when you run it. It will run only like donkey.

Now back to previous code (virtual classes topic), base pointer holding derived class object. So as our point" function should be called based on the object (or) not on pointer? It should run the object's function. But in C++, unfortunately it calls the base class function? OMG! This is logically wrong. So in C++, it doesn't happen.

If you call the donkey as a horse and say run, it will run like a horse. That's meaning in C++.

How to solve this?

C++ gives the keyword "virtual"

```
int main()
{
Base *p = new derived();
p->func();
}
```

```
class Base
{
public:
virtual void fun()
  {
  cout << "fun of Base";
  }
};
class Derived : public Base
{
public:
  void fun()
  {
  cout << "fun of Derived";
  }
};
```

205. Polymorphism:-

```
class Car
{
  public:
virtual void start()
  {
  cout << "Car started";
  }
virtual void stop()
```

```
    {
    cout<<"Car stopped";
    }
};

class Innova : public Car
{
 public:
   void start()
   {
    cout<<"Innova started";
   }
   void stop()
   {
    cout<<"Innova stopped";
   }
};
class swift : public car
{
 public:
   void start()
 {
  cout<<"swift started";
 }
 void stop()
 {
  cout<<"swift stopped"; } };
```

```
main()
{
  Car *C = new Innova();
  C->start(); → "Innova started" (Line no: 2)
  C= new Swift();
  C->start(); → "Swift started" (Line no: 4)
```

Here 2 & 4 are same crt? But different functions.
This is polymorphism.

Here we have innove car & Swift car are already existing.
Then we have written a class called Car. For what?
To achieve polymorphism. So that we call Innova as "Car" and Swift as "Car". That's why we have defined it.
All Classes having Common functions. Then why start() & stop() are there in class Car? Are you create object for class Car? No. So remove that.
After removing,

```
class Car
{
public:
 virtual void start();
 virtual void stop();
}
```

those are virtual functions. Yes. Do they have body? No. what is the purpose? To achieve the polymorphism. Why we don't define there? Because we want those functions implemented by Sub classes. you want to force derived class to implement those functions (or) just there? No, we want to force. what do we want to force? Any class that inherits from class Car, then it must override those two functions. How to make it Compulsory?

we have to write,

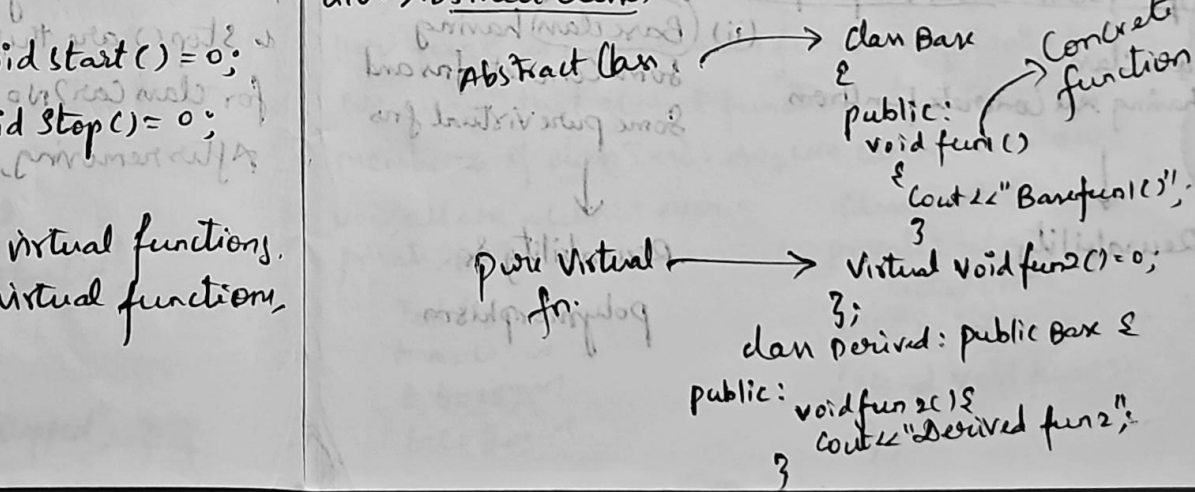
```
class Car
{
public:
virtual void start() = 0;
virtual void stop() = 0;
};
```

Then functions are called as pure virtual functions. when we are assigning zero to virtual functions, it is pure virtual function.

What does that mean? Those functions must be overriding by their derived classes, otherwise derived classes become abstract. class Car is also abstract. We can't create object for that class. When class inherits from Car, that class also become abstract, if it is not overriding.

Conclusion:- pure virtual functions must be overrided by derived classes. And purpose of pure virtual function is to achieve the polymorphism. We can't create object of Car class, but we can create pointer of class Car.

210. Abstract classes



Abstract classes can't have objects, but we can have pointers to achieve polymorphism.

Base *b; ✗ → we can't.

Base *p = new derived(); ✓ → yes we do.

p->fun2();

↳ These two statements leads to and achieve the polymorphism.

Two usage of Inheritance:-

1. To share features to child classes.
2. To achieve polymorphism.

3 types of classes Based on purpose :-

(i) Base class having All Concrete functions

↓

Reusability

(ii) Base class having some Concrete fns and some pure virtual fns (Abstract)

↓

Reusability & polymorphism

(iii) Base class having only all pure virtual fns

↓

polymorphism

} → Abstract class (or) If Everything is abstract, (i.e) We can call it as "Interface".

21.2. Friend function:-

```
void fun()
{
  Test t;  ✓
  t.a=10;  ✗
  t.b=15;  ✗
  t.c=20;  ✓
}
```

```
class Test
{
  private:
    int a;
  protected:
    int b;
  public:
    int c;
};
```

Here fun() is not member function of Test, but we want that global function to access the members of class Test. so, we can do as below,

Now, all are accessed even private, protected.

```
Test t;
t.a=10; ✓
t.b=15; ✓
t.c=20; ✓
```

```
class Test {
private: int a;
protected: int b;
public: int c;
friend void fun();
};
```

Friend Class:-

```
class your
{
public: my m;
void fun()
{
  ✗ Cout<<m.a; → we can't access private member of
}                 class from other class. Because
                  it is not inherited. (even inherited,
                  we can't access private ☺).
```

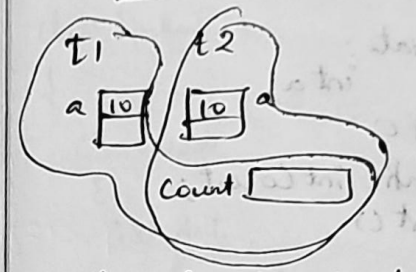
we can do as below,

```
class your
{
public: my m;
void fun()
{
  Cout<<m.a; ✓.
}
```

```
class my
{
  private: int a=10;
}
```

```
class your; → necessary.
class my
{
  private: int a=10;
}
```

15. Static member:-



Count is shared by objects and occupy only one space.

```
int main()
{
  test t1, t2;
  Cout<<t1.Count; → 2
  Cout<<t2.Count; → 2
  also → Cout<<test::Count; → 2
}
```

```
class Test{
private: int a;
public:
 static int Count;
 Test()
 { a=10;
  Count++;
 };
```

→ int Test::Count=0; must be written.

Static function:-

Static function in a class can access only static members, it can't access non-static member.

```
class Test
{
  private:
      int a;
  public:
  static int Count;
  Test()
  {
    a=10;
    Count++;
  }
  static int getCount()
  {
    a++; X
    return Count;
  }
};
int Test::Count=0;
```

```
int main()
{
Cout<< Test::getCount();   → return 0
Test t1;
Cout<< t1.getCount();      → return 1.
}
```

Non-static member can't be accessed by static member functions.

219- Inner/Nested class:

To reduce Complexity of bigger class.

Inner class can't access members of outside except static. (:-) Inner class can access static member of outer.

```
class LinkedList
{
  class Node
  {
    int data;
    Node *next;
  };
  ≡
};
```

```
class outer {
public:
  ① int a=10;
  ② static int b;
    void fun()
    {
      i.show(); ↵
      Cout<< i.x; ↵
    }
  class Inner
  {
  public:
    int x=25;
    void show()
    {
      Cout<<b; ↵
    }
  };
  ③ Inner i;
};
int outer::b=20;
```

but `Cout<<a;` X

↓

bcoz a is non static member.

221. Exception Handling:-

Error:-

programmer
- 1. Syntax Error → Compiler
- 2. Logical Error → Debugger
- 3. Runtime Error → (i) Bad I/P, (ii) problems with Resources.

⇓

These are called exception errors.

What is objective of exception Handling?

Giving proper message to the user, informing about the exact problem and also providing him the guidance to solve that problem.

222. Exception Handling Construct:-

```
int main()
{
  int a=10, b=0; c;
  try
  {
    if (b==0)
      throw 2;   // any value
    c=a/b;
    cout<<c<<endl;
  }
  catch(int e)
  {
    cout<<"Division by zero"<<"error code"<<e<<endl;
  }
  cout<<"Bye";
}
```

224. Throwing & Catching b/w funcs:-

```
int main()
{
  int a=10, b=0, c;
  try {
    c= division(a,b);
    cout<<c;
  }
  catch(int e)
  {
    cout<<"Division by zero"<<"errorcode"<<e;
  }
  cout<<"Bye";
}
```

```
int division(int x, int y)
{
  if (y==0)
    throw 1;
  return x/y;
}
```

226. All about throw:

We can throw integer, float, double, character and even class also.

Eg: Class myexception: public exception → build in function
{
}
int division (int x, int y) throw (myexception)
{
if (y==0)
throw 1;
return x/y;
}

We can write (or) need not.

Eg: int division (int x, int y) throw ()
{
=
if (y==0)
throw 1; → Gives warning, but it will throw actually
return x/y;
}

It means it doesn't throw any error.

228. All about Catch:

1. We can have multiple catch blocks.

```
try {
    1 — int
    2 — int
    3 — my Exception
    4 — float, char
}
Catch(int a)
{ -
}
Catch(my Exception e)
{ --
}
Catch( ... )
{
}
```

This is type. Now it Catch all type of data type that actually threw

But If we do as below

```
Catch (...)
{ --
}
Catch (int)
{
=
}
Catch (my Exception)
{ -
}
```

Here all will come under Catch(...) so, Catch(...) must be used last.

It is better to give Catch separately than giving Catch(...)

nesting:

```
try {
    ==
    try
    {
    ==
    }
    Catch()
    {=
    }
}
Catch()
{=
=
}
```

Other example:

```
class myexception1
{
};
class myexception2: public myexception1
{
};
```

```
try
{
==
}
Catch(myException2 e)
{
}
Catch(myException1 e)
{
}
```

→ Child should be first. Because exceptions are stored in hierarchy.

23). Template functions and classes:-

Eg:

```
template <class T>
T maximum(T x, T y)
{
 return x>y?x:y;
}
maximum(10,15);
maximum(10.5f,20.3f);
```

Eg:

```
template <class R, class G>
void add (R x, G y)
{
  cout<<
  return x+y;
}
int main()
{
  add (10.5, 2);
```

Most powerful feature in C++ is template:

```
template <class T>
Class stack
{
 private:
   T s[10];
   int *top;
 public:
   void push (T x);
   T pop();
};
```

```
template <class T>
Void stack <T> :: push (T x)
{
  =
  =
}
template <class T>
T stack <T> :: pop()
{ =
}
int main
{
  stack <int> s; → Creating stack of integer type data
  stack <float> s; → Creating stack of float
}
```

233. Constant Qualifier:

Const int x=10; → Can't modify data

int Const *p=&x; → Can't modify address

Const int Const *p=&x → Can't modify data & address

Eg: class demo

```
{
 public:
    int x=10;
    int y=20;
    void display()
    {
      x++;
      Cout<< x <<" " <<y<<endl;
    }
};
int main()
{
   demo d;
   d.Display();
}
```

If we don't want for display to modify the members of class, we can call below

```
Class :
  - -
  - -
  void display() Const   <- Compiler tell when we modify members.
  {
    x++; X
    - - -
    - - -
  }
```

Eg:- Call by reference (means act as inline / in Code get replaced).

```
 void fun(Const int &x, int y)
 {
   x++; X
   Cout<< x <<" " <<y;
}
int main
 {
   int a=10, b=20;
   fun(a,b);
 }
```

235. preprocessor directives/Macro.

237. Namespaces :-

```
namespace first
{
 void fun()
  {
   Cout<<"first";
  }
}
```

```
namespace Second
{
 voidfun()
   {
    Cout<<"Second";
   }
}
```

```
int main()
{
    first::fun();
    second::fun();
}
```

(or)

```
using namespace first;
int main()
{
    fun();
    second::fun();
}
```

Likewise cin, cout are in namespace of std.
Thatswhy we use using namespace std;.
We can also group all classes and related functions in a namespace and use it in the code.

239. Destructor:-

use of Constructor → ① Initialize them
② allocation of resources

use of Destructor → ① Deallocating & freeing of resources

Example:-

```
class Test
{
    int *p;
    ifstream fis;
public:
    Test()
    {
        p = new int[10];
        fis.open("my.text");
    }
    ~Test()
    {
        delete[]p;
        fis.close();
    };
```

We should not have multiple destructor. It can be virtual also.

Eg:-

```
class Demo
{
public:
    Demo()
    {
        cout << "Constructor of Demo" << endl;
    }
    ~Demo()
    {
        cout << "Destructor of Demo" << endl;
    }
};
void fun()
{
    Demo *p = new Demo();
}
int main(){ fun(); }
```

o/p:
Constructor of Demo
→ is not executed.

Because when we create dynamically, it is not freed / de-allocated automatically, so,

```
void fun()
{
  Demo * p = new Demo();
  delete p;
}
```

⟶ If we do then o/p.

o/p:
Constructor of Demo.
Destructor of Demo

24) Virtual destructor:-

```
class Base
{
public:
   Base()
   {
    cout << "Base Constructor" << endl;
   }
  ~Base()
   {
   cout << "Base destructor" << endl;
   }
};
```

```
class Derived :: public Base
{
 public:
    Derived()
    {
     cout << "Derived Constructor" << endl;
    }
  ~Derived()
   {
    cout << "Derived Destructor" << endl;
   }
};
int main()
{
  Derived d;
   .
   .
   .
}
```

⟶ o/p:

Base Co...
Derived Con...
Derived des...
Base dest...

Same example but with different main

```
int main()
{
 Base *p = new Derived(); → o/p:  Base Constructor
                                  Derived Constructor
 :
 Delete p;        ⟶  o/p:-  Base destructor.
}
```

Here C++ always see pointer object not the object attached.

Suppose if we want to also call destructor of derived.

We have to use virtual.

(ie)

```
virtual ~Base()
        {
        }
};
```

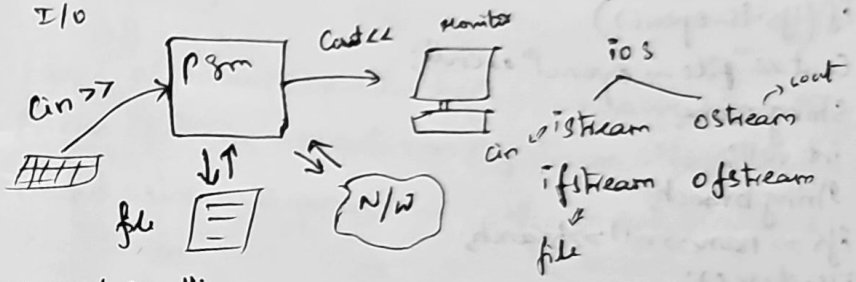
So,

```
int main()
{
    Base *p = new Derived();
    :
    delete p; →
}
```

o/p: Base Constructor
Derived Constructor
Derived destructor
Base destructor

243. Streams :-

I/O



244. File Handling :-

```
#include <fstream>
int main()
{
ofstream outfile("my.txt");
Outfile << "Hello" << endl;
Outfile << 25 << endl;
Outfile.close();
}
```

ios::trunc → To truncate

ios::append

Eg:

```
ofstream outfile("my.txt", ios::trunc);
```

245. Reading file:

```
int main()
{ ifstream ifs;
  ifs.open("my-txt");
if (ifs.is_open())
cout << "File is opened" << endl;
string name;
int roll;
string branch;
ifs >> name >> roll >> branch;
ifs.close();
}
```

248. Serialization :- → procen of string & retrieving state of an object, → class must have default Construction

```
class student
{
private:
    string name;
    int roll;
    string branch;
public:
    student() {}
    student(string n, int r, string b)
    {
     name=n;
     roll=r;
     branch=b;
    }
friend ofstream &operator<<(ofstream &ofs, student s);
friend ifstream &operator>>(ifstream &ifs, student &s);
friend ostream &operator<<(ostream &os, student &s);
};
ofstream &operator<<(ofstream &ofs, student s)
{
  ofs << s.name << endl;
  ofs << s.roll << endl;
  ofs << s.branch << endl;
  return ofs;
}
ifstream &operator>>(ifstream &ifs, student &s)
{
  ifs >> s.name;
  ifs >> s.roll;
  ifs >> s.branch;
  return ifs;
}
```

```
ostream &operator<<(ostream &os, Student &S)
{
 os<<"Name"<<S.name<<endl;
 os<<"Roll"<<S.roll<<endl;
 os<<"Branch"<<S.branch<<endl;
 return os;
}
int main()
{
 ofstream ofs("Test.txt");
 Student S1("John", 10, "CS");
 ofs<<S1;
 Student S2;
 ifstream ifs("Test.txt");
  ifs>>S2;
 Cout<<S2;
}
```

24. File Handling :-

Text file :- (Human readable)

Let say if we write 23 in text file,

2 3 ⇒ so in Text file,

ASCII 50, ASCII 51

00110010 00110011

→ For every instance, it is converted into ASCII and place it in file. So it is slow.

→ File size is also bigger.

Binary file (Machine file)

23 → Convert to binary 00000...0010111

mode: ios::binary

read();

write();

25. Manipulators :-

→ used for enhancing / formatting strings, data types.

```
Cout<<endl;
Cout<<"\n";
Cout<<hex<<163;
Cout<<fixed<<125.731;
Cout<<set(10)<<"Hello";
Cout<<10<<ws<<2;
```

integer: hex, oct, dec

float: fixed, scientific

other: set() → Hello (span), left → align, right → align, ws → whitespace

252. Why STL ? (Standard Template Library)

→ Data structure is used to store / manage the data.

→ If we creat arrays in static, Insertion of element needs the additional size so space is problem.

→ TO Solve that, create arrays in dynamically and if we want additional size, create new & copy all old data from previous array

253. Types of Data structures :-

① Linked List - Single / Double ② Stack ③ Dequeue ④ map

There is already built-in class available for above data structures (is) called STL.

254. STL :

It Contains Algorithms, Containers, Iterators

① Algorithms for → Search(), Sort(), Binary-search(), reverse(), Concat(), Copy(), union(), Intersection(), merge(), Heap().

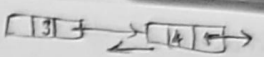
② Containers → any datatype.

All the below classes are template classes.

① Vector :- → Self-managed dynamic array (automatically

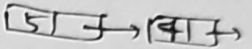
Push-back(), pop-back(), Insert(), remove(), size(), Empty()

② List → Double Linked List (i)



[vector functions + push-front(), pop-front(), front(), Back().

③ Forward-List :



Same functions.

④ Dequeue :- Same as vector but at both end, we can Insert / delete.

⑤ priority-Queue → Heap data structure, remove, add → High priority elements

⑥ Stack → LIFO

⑦ Set → Contains only unique elements.

⑧ Multiset → Same as Set but allows the duplicate.

⑨ map → By Hash table

useful for storing <Key ; Value> pair.

Eg: 1 "John"
2 "Ajay"

255. STL Classes :-

③ Iterators

↳ used for elements through Containers.

```
#include <vector>
main()
{
  vector<int> v={10,20,30,40};
  v.push_back(25);
  v.push_back(70);
  vector <int> :: iterator itr;
  for (itr= v.begin(); itr!= v.end(); itr++)
      cout<< *itr;
}
```

257. Map classes :-

```
#include<map>
int main()
{
 map<int, string> m;
 m.insert (pair<int,string> (1,"John"));
 m.insert(pair<int, string> (2, "Ravi"));
 map <int, string> :: iterator itr;
 for(itr=m.begin(); itr!=m.end; itr++)
 {
   cout << itr -> first << " " << itr -> second << endl;
 }
```

```
 map <int,string>:: iterator itr1;
 itr1 = m.find(2);
 cout << "Value found in" << endl;
 ~~itr1 = m.find(2~~
 cout << itr1 -> first << " " << itr1 -> second << endl;
}
```

259. Auto :-

```
int main()
{
  auto x= 2* 5.7+'a';
  cout<<x;
}
```

→ automatically Compiler find the data type (biggest size data type)

260. Final Keyword :-

It used to restrict the inheritance as well as overriding all functions.

Eg:-

```
class parent
{
};
class child: parent
{
};
```

works But if we

```
class parent final
{
};
class child: parent
{
}
```

error arise. (it) restricted.

Eg2:
```
class parent
{
   virtual void show() final
   {
   }
};
class child: parent
{
  void show()
  {
  }
};
```
← Even if we make virtual, show() is not overrided. Because final is used.

26). Lambda expression:-

[capture-list](parameter_list) → return type { body };

```
Eg:
main()
{
Auto f = [](){Cout << "Hello";};
[](int x, int y){Cout << "Sum:" << x+y;}(10,5);
int x = [](int x, int y){return x+y;}(10,5);
f(); (we can also call)
```
(f → reference)

Eg2:
```
main()
{
 int a=10;
 int b=5;
[](){Cout << a << " " << b;}(); → error a,b is undefined. Because it is nested function
```
We can do,
```
[a,b](){Cout << a << " " << b;}();
```
To modify the Content,
```
[&a,&b](){Cout << ++a << " " << ++b;}();
```
(or)
```
[&a](){Cout << ++a << " " << ++b;}();
```
→ means we can access all values

If we want return types,
```
int a = [](int x, int y) -> int {return x+y;}(10,30);
Cout << a;
```

263. Smart pointers :-

Problem of pointers:

```
void fun()
{
  Rectangle *p = new Rectangle();
  ..
  ..
  delete p;   <- If we don't do it, p is not deleted,
}             (i.e) memory leak -> Cause heap memory
              get fully overflowed -> crash happens.

int main()
{
  while(1)
  { fun();
  }
}
```

C++ provides Smart pointers which automatically delete once out of the Slope.

e.g.

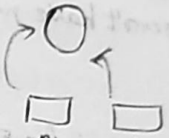
1. unique_ptr
2. Shared_ptr
3. Weak_ptr

1. unique_ptr Example:

```
fun()
{
unique_ptr<Rectangle> p1(new Rectangle(10,5));
 Cout << p1->area();
 Cout << p1->perimeter();
}   <- goes out of slope, automatically deleted.

int main()
{
  while(1)
  { fun();
  }
}
```

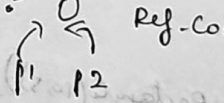
unique_ptr:-



Here only one pointer

If we are using a unique pointer, then if one object is created and pointer p1 is pointing to that object only one pointer can point on this object. we can't share with another pointer. But we can transfer control to p2 by removing p1.

Shared pointer:-



Weak - pointers :- Same as shared pointer but it doesn't maintain reference Counter. So pointing will be there - It is Weak. Being pointer will not have strong hold on the Object. The reason is if suppose pointer are holding the objects and requesting for other objects then they may form deadlock b/n pointers. So to avoid a deadlock, Weak pointers are useful, so it doesn't have ref-Counts.

Eg: int main()
{
unique_ptr<Rectangle> ptr(new Rectangle(10,5));
Cout << ptr->area() << endl;
unique_ptr<Rectangle> ptr2;
ptr2 = move(ptr);
Cout << ptr2->area();
Cout << ptr->area();
}

O/P:
50
50
→ Null (Because ptr is not having Rectangle object)

Eg: int main()
{
shared_ptr<Rectangle> ptr(new Rectangle(10,5));
Cout << ptr->area() << endl;
shared_ptr<Rectangle> ptr2;
ptr2 = ptr;
Cout << "ptr2 " << ptr2->area() << endl;
Cout << ptr.use_count() << endl;
}

O/P:
50
ptr2 50
ptr 50
2.

265. In class Initializer and Delegation of Constructor :-

class Test
{
int x=10;
int y=13;
public:
Test(int a, int b)
{
x=a;
y=b;
}
Test(): Test(1,1) → one Constructor is Calling the other Constructor.
{
}
};

266. Ellipsis

It is used to pass variable no. of arguments to the function.

Sum(5,6,7);
Sum(5,9,10,11,13);
→ for variable no. of arguments, it will work.

int Sum()
{
}

```
int Sum(int n, ...)
{
    va_list list;
    va_start(list, n);
    int s=0;
    for(int i=0; i<n; i++)
      s+= va_arg(list, int);
    va_end(list);
    return s;
}
Sum(3, cout<< Sum(3,10,20,30);
    cout<< Sum(7, 5, 2, 3, 4, 5, 6, 7);
```

Linux Commands

/ => "Root", the top of file system hierarchy.

ls, cd, cat, pwd, echo, man, exit, clear

ls -l

Which Command Exactly?

[root@svr ~]$ which cat

/bin/cat

Directory shortcuts

. This directory

.. The parent directory

cd - change to previous directory

mkdir, rmdir, rm -f [Recursively removes directory]

Decoding ls -l output

$ ls -l

ls -t => List files by time, ls -s, ls -R

↓ Recursively

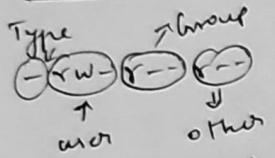
=> Consider same,

Symbol	Type
-	Regular file
d	Directory
l	Symbolic link

Symbol r -> Read,

Symbol u -> user, g -> Group, o -> other, a -> All

Secret decoder Ring



Changing permissions :-

Item	Meaning
chmod	change mode command
ugoa	User Category, user, group, other, all
+-=	Add, Subtract (or) set permissions
rwx	Read, write, Execute

Removing files

rm file, rm -r dir, rm -f file => force removal & never prompt for Confirmation.

Copying :-

cp Source-file destination-file

Copy Source-files to destination file:-

cp srcfile1 srcfile2 ... dest-dir

mv Source destination

sort file => Sort text in a file

Creating a Collection of files :-

tar [-] c|x|t f tarfile [pattern]

Eg: tar -c date.sh a.sh f.sh -f (all)

↑ This is tarfile name

C => Create tar archive, x => Extract files from archive,

t => Display table of Contents

Compress file

gzip, gunzip → uncompress

@ $ ~script/ls

all.sh date.sh

@ ~script $ gzip all.sh date.sh

@ ~script/ls

all.sh.gz date.sh.gz

@ ~script$ ls gunzip all.sh.gz date.sh.gz

@ ~script $ ls

all.sh date.sh.

I/O redirection:-

o/p direction

Cat (>) filename → overwrite

Cat (>>) filename → append

Eg: Cat > simple.txt

```
a
b
c
d
^c → exit.
```

Cat simple.txt

```
a
b
c
d
```

Cat > simple.txt

```
e
f
g
h
^c → exit
```

Cat simple.txt

```
e
f
g
h
```

Eg: Cat simple.txt

```
e
f
g
h
```

Cat >> simple.txt

```
L
m
```

Cat simple.txt

```
e
f
g
h
L
m
```

Cat <<EOF> file.txt

```
a > a
  > b
  > c
  > EOF
```

which we can enter i/p until a certain sequence (EOF) is typed.

Linux

① What is Linux :-

→ An OS, a Kernel (heart of os), Created by Linus in 1991

→ first version released in 1994.

→ FOSS (Free/open Source Software)

→ Unix-Like, → Linux is layer between applications & Hardware

Linux Distributions :-

→ Linux Kernel + Additional Software

→ Each distribution can have different focus

→ Many distributions available to choose from

DistroWatch.com → Website to learn Linux distributions.

Lists :-

① Red Hat Enterprise Linux ② Fedora ③ Ubuntu ④ Debian

⑤ SUSE Linux Enterprise Server (SLES) ⑥ OpenSUSE

Why Linux?

→ Runs on many hardware platforms.

But let's say for Unix, Hp-UX only runs on Hp servers, AIX only runs on IBM servers,

Linux can run on Hp, IBM and other servers.

→ Small footprint.

So, Small footprint of Linux allows it run on older hardware (or) Embedded systems

→ Stable, Reliable, Secure

→ Great for Servers

③ Linux distributions:-

Redhat

Popular in

→ Banks, Airlines, Telecoms, Healthcare

Having pay License.

But Centos is a Red Hat Enterprise Linux derivative that's Completely free

Ubuntu:

popular in

→ startups, Saas, Social networks, Cloud Based

→ Linux Concepts are universal

→ Each distro is slightly different

→ You can accomplish the same goals on most Linux distros

→ you can't make "wrong" choice!

Linux Distro = Kernel + Software

CentOS = RHEL — branding / logos

Common directories

/ "Root", top of the file system hierarchy.

/bin Binaries and other executable programs.

/etc System configuration files

/home Home directories.

/opt optional (or) third party software

/tmp Temporary space

/usr user related programs

/var Variable data,

Shell → Command Line Interface to Linux operating sys. (CLI):-

what is shell → ① Default Interface to Linux ② A pgm that accepts our commands & execute those commands ③ Also called a Command Line interpreter

CLI vs GUI:-

→ CLI is more powerful, there will always be Command line.

→ Server distributions do not include GUIs, → Desktop distributions have GUIs and CLIs

I/O redirection:-

Cat

Eg: C
a
b
c
d
^c

Cat s
a
b
c
d

Cat >
e
f
g
h
^c

Cat si
e
f
g
h

Cat <

a > a
> b
> c
> EOF

The prompt

[jason@linuxsvr ~]$ ⇒ Gives us some sort of information.

- jason → user name
- linuxsvr → name of the linux system
- ~ → ?
- $ → dollar sign ⇒ Indication that we are using system as normal user as opposed to superuser.

[root@linuxsvr: ~]# ⇒ superuser shell point ends with pound sign.

Superuser on linux system is also called root.
It is root account.

Root, the Superuser:-

→ Root is all powerful
→ Normal accounts can only do subset of things
root can do.

Tilde expansion

~jason = /home/jason
~pat = /home/pat
~root = /root
~ftp = /srv/ftp

Basic Linux Commands

→ ls → Lists directory contents
→ cd → changes the current directory
→ pwd → Displays the present working directory
→ cat → Concatenates and display files
→ echo → Displays arguments to the screen.
→ man - Displays the online manual.
→ exit → Exits shell or our current session.
→ clear → clears the screen.

Teaching oursel f

→ Man is used to display documentation.
→ $path Controls your search path
(Capital letter)
→ Learn the full path to Commands with which
→ ASK Commands for help with --help or -h.
→ Search man pages by using man -k.

Working with Directories:-

Directories

→ Are Containers for other files & directories
→ provide a tree like structure
→ Can be accessed by name (or) shortcut

Directory shortcuts

- . → This directory
- .. → The parent directory

cd - → change to the previous directory

Creating and Removing Directories

mkdir [-p] directory - Create a directory

rmdir [-p] directory - Remove a directory

rm -r directory - Recursively removes directory

Listing files and Understanding ls o/p:-

$ ls -l → shows folders (or) files with full details

$ ls -la (or) ls -al → show full details along with hidden files.

$ ls -a → show all files with hidden files

Listing files by Type:-

use ls -F to reveal file types

/ Directory

@ Link

* Executable

ls -t → list files by time

ls -r → Reverse order

ls -latr → long listing including all files reverse sorted by time.

ls -R → lists files recursively

The tree Command:-

Similar to ls -R, but creates visual output.

tree -d → list directories only

tree -C → Colorize output

list directories, Not Contents

ls -d → List directory name, not contents

ls --colour → Colorize the output

Working with spaces in Names

→ Just say no spaces

→ Alternatives: -, _ → Hyphen → underscore.

permissions

Symbol	Type
-	Regular file
d	Directory
l	Symbolic link

Symbol	permissions
r	Read
w	write
x	Execute

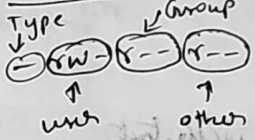
Permissions	File	Directory
Read (r)	Allows a file to be read	Allows file Names in the directory to be read
Write (w)	Allows a file to be modified	Allows entries to be modified within the directory
Execute (x)	Allows the execution of a file.	Allows access to contents and metadata for Entries.

Symbol	Category
u	User
g	Group
o	other
a	All

Groups :-

→ Every user is in atleast one group.
→ Users can belong to many groups.
→ Groups are used to organize users.
→ groups Command display a user's groups.
→ you can also use id -Gn.

Secret Decoder Ring:



1 bob users 10400 Sep 27 08:52 Sales.data

Change permissions

chmod → Change Mode Command
ugoa → user Category user, group, other, all
+= → Add, Subtract (or) Set permissions
rwx → Read, Write, Execute

Eg: - --- $ ls -l vicky.txt

-rw- rw- r-- 1 wdlcome wdlcome 20 oct 30 19:00 vicky.txt

owner (i.e) user has read and write permission

group

To remove read access for us (owner/user),

$ chmod u-r vicky.txt

$ ls -l vicky.txt

--w-rw-r-- 1 wellcome wellcome 16 oct 30 19:02 vicky.txt

Now go & check the file in folder. ⊠ vicky.txt

"Could n't open the file". => error

Now try to give read access.

$ chmod u+r vicky.txt

$ ls -l vicky.txt

-rw-rw-r-- 1 wellcome wellcome 16 - - - vicky.txt

Sprb: ☺ - Linux is excellent.

If we do,

$ chmod a=r vicky.txt

$ ls -l vicky.txt

-r--r--r-- 1 wel - - - => all are set read

Commonly used permissions

Symbol	octal
-rwx------	700
-rwxr-x-r-x	755
-rw-rw-r--	664
-rw-rw----	660

Working with Groups

- New files belong to your primary group.
- chgrp command changes the group.

Directory permissions Revisited

- permissions on a directory can affect the files in the directory
- If file permissions look correct, start checking directory permissions.

File creation Mask :

- file creation mask determines default permissions
- If no mask were used, permissions would be :
- 777 for directories
- 666 for files

Displaying the Contents of files

Cat file	Display the Contents of file
more file	Browse through a text file.
less file	More features than more.
head file	Output the beginning (or top) portion of file.
tail file	Output the ending (or bottom) portion of file.

Head & Tail:-

→ Displays only 10 lines by default
→ Change this behavior with -n
→ n = no. of lines
→ tail -15 file.txt

Viewing files in Real time

tail -f file → follow the file } → for log files
Displays data as it is being written to the file.

→ Do the experiment in terminal.
Ex: $ Cat vicky.txt ——→ write 20 lines already.
:
:
$
→ $ head -15 vicky.txt → shows first 15 lines
:
:
→ $ tail -3 vicky.txt → shows last 3 lines
:
:

Vi editor:-

→ More advanced than nano.
→ Vi has three modes

$ vi vicky.txt → it opens file in vi editor
$ vimtutor → open tutorial page → find & download a cheat code
Emacs Editor → powerful editor

Two commands that allow us to find files & directories

① Find Command

find [path ...] [expression]
→ Recursively finds files : path that match expression. If no arguments are supplied it find all files in the current directory

find options

-name pattern	Find files & directories that match pattern
-iname pattern	Like -name, but ignores Case.
-ls	perform an ls on each of found items.
-mtime days	Finds files that are days old.
-size num	Finds file that are of size num.
-newer file	Finds file that are newer than file.

-exec Command {} \;
Run Command against all the files that are found.

$ find → find all files

$ find -name vicky.txt → find that file

$ find -name VICKY.txt ⇒ doesn't find file because Case sensitive.

$ find -iname VICKY.txt ⇒ find the file vicky.txt

$ find size +200M ⇒ show file that has size more than 200Mb.

② A Fast Find - locate

locate pattern
→ Lists files that match pattern
→ Faster than the find command.
→ Queries an index.
→ Results are not in real time

locate command however works off of an index. Typically index is rebuilt everyday. Instead of Evaluating each & every file to see if it matches the pattern, locate just does a lookup in index. Results are not in real time. There is lag b/n time that index gets created and that you run the locate command.

So, if you are looking for a file that was created just two (or) three minutes ago, chances are it's not going to be in index. So we have to use "find" command

Graphical Editors

- emacs → Emacs has a graphical mode too.
- gedit → The default text editor for Gnome.
- gvim → The graphical version of vim.
- kedit → The default text editor for KDE
- AbiWord → Microsoft Word alternative
- Libreoffice → Full office suite
- kate → Source code editor for programming

Removing files

rm file Remove file

rm -r dir Remove the directory and its contents recursively.

rm -f file force removal and never prompt for confirmation.

Copying files

cp source-file destination-file ⇒ Copy source-file to dest.

cp src-file1 [src-file2 ... src-fileN ...] dest-dir

↓

Copy source files to destination-directory.

cp -i

Run interactive mode.

cp -r source_directory destination.

Copy src-directory recursively to destination.

Eg: tx1 tx2

Hi, vignesh1 Hi, vignesh2.

$ cp tx1 tx2 => Copied into tx2.

$ cat tx2

Hi, vignesh1 => (tx) Copied.

Eg2:- tx1 tx2

vignesh1 vignesh2

$ diff tx1 tx2

Answer:

1c1

< vignesh1

> vignesh2

Eg3: $ rm tx1 => remove file

Eg4: $ cp tx1 tx2 dir1 => Copy tx1, tx2 into "dir1" directory

assume 'dir1' is already created.

Eg5: $ rm -r database => Remove directory 'database'

Moving & Renaming files:-

mv Move (or) rename files & directories

mv source destination

mv -i source destination.

Sorting Data

Sort file => sort text in file

Sort options

-k F Sort by key. F is the field number.

-r Sort in reverse order

-u Sort unique

Eg: data => txt file contains

User: vignesh
Emp: 3422416
Loc: COB
Dept: EKI-XC
Level: 51
Team: Audio

$ Sort data => Sort them in ascending letter.

$ Sort -r data => Sort them in reverse.

Creating a Collection of files:

tar [-] c|x|t f tarfile [pattern]

⇒ Create, extract (or) list contents of tar archive using pattern, if supplied.

tar options:

c → create a tar archive.
x → ~~creat~~ Extract files from the archive.
t → Display the table of contents (list).
v → Be verbose.
z → Use compression.
f file → use this file.

Eg: $ ls

Desktop documents (test) ⇒ directory contains txt1, txt2 files.
Music . . .

$ tar cf new.tar test ⇒ Create an archive which has name "new.tar" & contains the files/folders of test directory

$ ls

Desktop documents (new.tar) test.
Music . . .

Now trying to extract files in new directory.

$ mkdir temp

$ ls

Desktop documents new.tar test
. . . (temp).

~~$ tar xf /home/welcome~~

$ cd temp ⇒ ~~moved~~ go to temp directory

$ tar xf /home/welcome/new.tar.

$ ls

test ⇒ here we can see "test folder" comes in (temp) directory

$ ls -l test
-rw-rw-r-- . . . txt1
-rw-rw-r-- . . . txt2

Just go to home (~) cd -

$ tar tf new.tar

test/
test/txt2
test/txt1

⇒ Now it lists.

Compressing files to Save space

gzip ⇒ Compress files
gunzip ⇒ UnCompress files
gzcat ⇒ Concatenates Compressed files
zcat ⇒ Concatenates Compressed files

Disk usage

du ⇒ Estimate file usage
du -k ⇒ Display size in Kilobytes
du -h ⇒ Display size in human readable format

WildCards

→ A character (or) string used for pattern matching.
→ Globbing expands the wildCard pattern into list of files and/or directories. (path)
→ WildCards can be used with most Commands
→ ls
→ rm

WildCards - 2 types

① * → asterik matches zero (or) more characters
Eg: *.txt → find file with .txt extension
Eg: a* → find file with Letter starting 'a'.
Eg: a*.txt → find " " with txt extension

② ? - matches exactly one character
?.txt → preceding any character with .txt
a? → To match all the two letters files that begin with an 'a'
a?.txt → To match all files starting 'a', then followed by exactly one more character and then end in .txt

More WildCards - character classes :-

• [] → A character class.
→ Matches any of the characters included between the brackets. Matches exactly one character
Eg: ① [aeiou]

② Ca[nt]*
Ca → started with 'Ca'
[nt] → followed by n (or) t
* → followed by zero (or) more characters
so, it can search Can, Cat, Candy, Catch.

[!] → Matches any of the characters NOT included b/n brackets. Matches exactly one character.

[!aeiou]*

So, it can find baseball, Cricket → because, they are not started with a, e, i, o, u.

More wildCards - Range

→ use two characters separated by a hyphen to create a range in character class.

[a-g]* ⇒ Matches all files that start with a, b, c, d, e, f, or g.

[3-6]* ⇒ Matches all file that start with 3, 4, 5 or 6.

Named character classes → also most common used (ie) predefined character classes.

[[:alpha:]] ⇒ alphabets

[[:alnum:]] ⇒ alphanumeric

[[:digit]] ⇒ digits

[[:upper]], [[:lower]], [[:space]]

Matching wildCard patterns

\ → escape character ⇒ use if you want to match a wildCard character.

Eg: Match all files that end with a question mark:

*\?

→ Eg: done!

Files:

a ab.txt b blues.mp3 c Cat e g (notes) → directory

aa a.txt bb b.txt Cat d f h songs.txt (music) → directory

$ ls ? ⟶ list single letter files

a b c d e f g h

$ ls ?? → list double letter file

aa bb

$ ls a?.txt

ab.txt

$ ls a*.txt

ab.txt a.txt

$ ls c[aeiou]t
Cat Cot
$ ls c[uoiae]t
Cat Cot
} → same results

$ ls [a-d]*

a aa ab.txt a.txt b bb b.txt c Cat Cot d

$ ls *[[:digit]]

blues.mp3

$ mv *.txt notes ⇒ move all txt files to notes

Input / Output Types

I/O Name	Abbreviation	File Descriptor
Standard I/P	stdin	0
Standard o/p	stdout	1
Standard Error	stderr	2

File descriptors are just numbers that represent open files for us. It's easier to reference files by name, but it's easier for Computers to reference them by number.

Redirection:-

The explicit way of using redirection is to provide a file descriptor number. If one is omitted, then file descriptor zero is assumed for input and one for output redirection.

> Redirects standard output to a file. overwrites (truncating) existing Contents.

>> Redirects standard output to a file. Appends to any existing Contents.

< Redirects input from a file to a Command

& used with redirection to signal that file descriptor is being used.

2>&1 Combine stderr and standard output.

2>file Redirect standard error to a file.

The Null Device

>/dev/null → Redirect output to nowhere.

If you want to ignore o/p, you can send it to the null device, which is dev null. The null device is a special file that throws away what ever is fed to it.

Watch lecture 31

Comparing the Contents of files:-

diff file1 file2 → Compare two files

sdiff file1 file2 => Side-by-Side Comparison
(file1 ⇓ Left, file2 ⇓ Right)

vimdiff file1 file2 => Highlight differences in Vim
(⇓ vim Editor)

diff Output :-

$ diff file1 file2

3c3
<

>

LineNumfile1 - Action - LineNumfile2

Action => (A) dd
(C) hange
(D) elete

Eg: $ diff file1 file2

3c3
< this is a line in file1 => separator
--- => line from file1
> this is a line in a file1 => line from file2

$diff output :-

$ sdiff file1 file2
line in file1 | line in file2

| => Differing lines
< => line from file1
> => line in file2

Stderr redirection :-

script$ y 2>>a.txt
script$ cat a.txt
y: command not found

Comparing files :-

diff file1 file2 => Compare two files
sdiff file1 file2 => Side-by-Side Comparison

Grep Command

It is generally used with pipe (|).

Eg: Cat Commands.txt | grep 9

Grep options :-

-i => perform a search, ignoring case.
-n => precede output with line numbers.
-c => Count the number of occurrences in a file.

cd Commands :-

cd ../../.., To go back one level, we can simply enter ".." (two dots) after cd command. What's cool, we can go back further by adding another two-dots and separating them with forward Slash (/).

find Command:

find [starting directory] [options] [Search term]

Starting directory:-

/ => Search whole system

. => Search the working directory

~ => Search the home directory

options:- => It is an additional argument that we can use to refine the Search.

-name => look for files based on names

-user => Search for files that bdong to a given user.

-size => look for files based on their sizes.

Search term: => Keyword (or) number that we use to Search for files.

Eg: find . -name "index"

What is SSH:

SSH (or) Secure Shell is a remote administration protocol that allows users to control & modify their remote servers over the internet. The service was created as secure replacement for the unencrypted Telnet & uses Cryptographic techniques to ensure that all Communication to and from the remote server happens in an encrypted manner. It provides a mechanism for authenticating a remote user, transferring inputs from the client to the host and relaying the output back to the client.

SSH Commands Consist of 3 parts:-

ssh {user}@{host}

ssh => Key Command instructs your system that you want to open an encrypted Secure Shell Connection.

user => represents the account we want to access. For example, we may want to access the root user, which basically synonymous for system administrator with Complete rights to modify anything on the system

host => refers to the Computer we want to access.

ssh root@10.120.1.91 → This can be an IP address (or) a domain name

diff Output :-

$ diff file1 file2

3c3

2c2

LineNumfile1 – Action – LineNumfile2

Action => (A) dd
(C) hange
(D) dete

Eg: $ diff file1 file2

3c3

< this is a line in file1

--- -> separator

> this is a line in a file1

< => line from file1

> => Line from file2

$diff output :-

$ sdiff file1 file2

line in file1 | line in file 2

| => Differing lines
< => Line from file1
> => Line in file 2

Vimdiff

Ctrl-W W Go to next window
:q => Quit (close current window)
:qa => Quit all (close both files)
:qa! => force Quit all

Eg: sdiff tx1 tx2

Name: Vignesh
Emp_ID: 34224125
DEPT: EKI-XC
LEVEL: 51
Team: Audio

(1) Name: vicky -> change
Emp_ID: 34224125
DEPT: EKI-XC
LEVEL: 51
(1) Team: Audioo -> change

Searching Contents of file

grep => Display lines matching a pattern

grep pattern file

grep options

-i performs a search, ignoring case.
-c Count the number of occurrences in file.
-n precede output with line numbers.
-v Invert Match. print lines that don't match.

txt ⇒ Contains Name : Vignesh
Emp-ID : 34224125
DEPT : ERI-XC
LEVEL : 51
Team : Audio

$ grep Emp-id txt
→ No o/p because Case sensitive
$ grep Emp-ID txt
Emp-ID : 34224125
$ grep (-i) Emp-ID txt → ignore Case-Sensitive
⇒ Emp-ID : 34224125

Searching for Text in Binary Files :-

Strings Display printable strings
When you use grep, it will check information was found in file, but it will not display the text.

Pipes

→ vertical bar
| ⇒ pipe symbol
Command - output | Command - input
⇒ pipes takes standard o/p from preceding Command and passes it as the standard i/p to the following Command.

→ If first Command displays error message, it will not passed to second Command by default.

pipes

grep pattern file
Cat file | grep pattern

The Cut Command

Cut [file] ⇒ Cut out selected portions of file. If file is omitted, use standard input.

Cut options

-d delimiter ⇒ Use delimiter as the field separator
-f N ⇒ Display Nth field

Transferring and Copying files over the Network

Copying files over the Network
→ scp - Secure Copy
→ (SFTP) - SSH file transfer protocol
↓
Secure file Transfer protocol

Inorder to use scp (or) sftp, we need a client.

Command Line scp clients

* scp } Mac & Linux
* sftp }
* PuTTY Secure Copy client - pscp.exe } Windows
* PuTTY Secure file Transfer client - psftp.exe }

Graphical SCP/SFTP Clients

→ Cyberduck
→ filezilla
→ WinSCP

SCP/sftp Command Line utilities

scp source destination ⇒ Copy source to destination

Sftp host ⇒ Start a secure file transfer session with host

Ex: $sftp vignesh-ravichandran

To know host name,

Ex: $ hostname
o/p: vignesh-ravichandran

Ex: $ sftp vignesh-ravichandran
Welcome @ vignesh-ravichandran's password: ← Enter password
Connected to vignesh-ravichandran.
sftp>

ftp Command Line utility :-

ftp host
⇒ start a file transfer session with host

Just be aware that FTP is not a secure transfer protocol like SCP and SFTP.

Ex: $ sftp vignesh-ravichandran
Connected to " ".
sftp> put txt
⇒ uploading txt into server
↓
automatically
Comes in screen

sftp > lpwd
o/p: Local Working directory : /home/Welcome
sftp > rm txt
Removing txt from server.
sftp > Quit.

Customizing the shell prompt

→ Use an Environment Variable to customize
→ Bash, ksh and sh use $PS1
→ CSh, tcsh and zsh use $prompt

To identify which shell is running:

$(ps -p $$) ⇒ Command

Customizing prompt with PS1

\d ⇒ Date in "Weekday Month Date" format
"TUesday 26".
\h ⇒ Hostname up to the first period
\H ⇒ Hostname
\n ⇒ Newline
\t ⇒ Current time in 24-hour HH:MM:SS format
\T ⇒ Current time in 12-hour HH:MM:SS format
\@ ⇒ Current time in 12-hour am/pm format
\A ⇒ Current time in 24-hour HH:MM format
\u ⇒ Username of the current user
\W ⇒ Current working directory
⇒ Basename of the current working directory
\$ ⇒ if the effective UID is 0, a # otherwise a $

Aliases

→ Shortcuts
→ use for Long Commands
→ use for Commands you type often

Creating Aliases

alias [name [=value]]
list (or) create aliases
Use name=value to create a new alias

Aliases:

→ used for fix typo error
$ alias grpe='grep'
→ Make Linux behave like another OS.
$alias cls='clear'

Removing Aliases

unalias name ⇒ Remove the "name" alias.
unalias -a ⇒ Remove all aliases

Eg. $ ls -l

: : :

Creating alias?

$ alias ll='ls -l'

So we can type

$ ll

: : :

$ cls

cls: Command not found.

$ alias cls='clear'

So we can type

$ cls

Learn - Git

1. what is version control?

Every project evolves some steps. changes here, then couple of changes there. Each of these steps are new version of the project. With version control, we can store each versions in like a database. Anytime later, we can see version & what change we did.

Without version control system :-

All are stored in folders. so some you are unresolved.

① what exactly was changed in each version?

② what do these changes mean? Did anyone comment them?

③ why do we store whole project, instead of just modification?

④ How do you keep variants of projects in sync?

Benefits :-

① Save versions properly ② Easily see and understand changes ③ Restore previous versions ④ Work safely in separate context ⑤ Collaborate without overwriting ⑥ Every team member is a backup.

② Installing & Configuring git

sudo apt install git

$ git config --global user.name "vicky"

$ git config --global user.email "vickycit9918@gmail.com"

$ git init

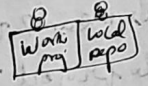
To help of any command

$ git init --help

③ Basic Workflow of Version Control

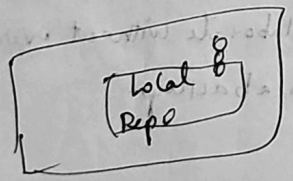
(A) Creating a Local Repository:

Local project that is not yet under VCS.



↓ new Local GitRepo for this project

(B) Cloning an Existing Remote Repository



clone

Existing repo on a remote server already under version Control

Local "clone" of this repo, downloaded to our local machine

working on your project's files:

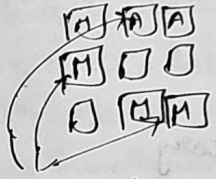
Work on your project's files (modify / add / delete / ...)

...ing any application (Editor, file Browser, IDE, ...)

Overview of changes:

Which files have you changed?, How exactly have you changed them?

Which changes shall be part of next Commit?



added to 'staging area'

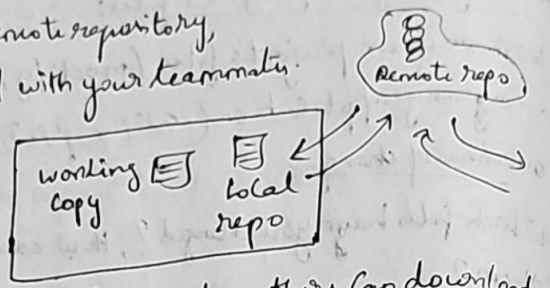
→ only staged changes are saved in local repository as a new Commit

Inspecting Commit history

The "log" shows how the project has evolved.

Sharing via Remote Repositories

[optional] Through a remote repository, data can be exchanged with your teammates.



we can submit new code to remote, others can download that & views

4. Creating a new local Repository :-

$cd projects
$git init -> it creates new empty repository
$ls -la
o/p:
. . . .git ⇒ folder ⇒ Don't mess with this folder, but we have to understand what it is.

5. Cloning a Remote repository :-

$git clone <url>
① ssh://user@server/git-repo.git
② user@server:git-repo.git
③ http://example.com/git-repo.git
④ http://user@- - -

$ cd ~/project
$ git clone https://github.com/jquery/jquery.git
o/p:
. . . .git → This contains lot of ~~git~~ commit history

6. Inspecting local changes

work in any application we like.
→ Modify, create, delete, rename, move...
nothing to watch out for in this step!
$ $ git add a.txt ——→ Hello this is vicky
$ git status
o/p: on branch main
No commits yet
Changes to be committed:
new file: a.txt

$ vi a.txt → Hello this is vicky
learning git

$ git status
o/p: on branch main
No commits yet
Changes to be committed:
new file: a.txt

changes not staged for commit:
modified a.txt

$ git diff a.txt

o/p:
diff --git a/a.txt b/a.txt
index ...
--- a/a.txt
+++ b/a.txt
@@ -1,2, +1,4 @@
This is vicky
+ learning git → green colour.

⑦ Staging & Committing changes :-

set of changes → .. saved as Commit → local repo

working copy
your project's files

staging Area

local repo
".git" folder

[M] tracked ... modified → [M] staged → Commited

[ ] tracked ... unmodified

[?] untracked
↳ new file

Eg: $ vi a.txt → Hi
$ git status

o/p: on branch main
untracked files:
(use "git add <...>")
a.txt
nothing added to commit but untracked files present.

$ git add a.txt
$ git status

o/p:
on branch main
changes to be committed:
new file: a.txt → staging area

$ git commit -m "new file added"

~~$ git status~~ o/p:
new file added
1 file changed, 1 insertion(+)
create mode 100644 a.txt

$ git status

o/p:
nothing to commit, working tree clean

$ vi a.txt => Hi vicky

$ git status

o/p: changes not staged for Commit:
modified: a.txt

$ git diff a.txt

o/p:
:...
-hi
+hi vicky

Compare

$ git add a.txt

$ git status

o/p:
Changes to be Committed:
modified a.txt.

$ git Commit -m"a file modified"

$ <--- That's all --->

if we modify, but if we Commit instead of add

Eg: $ git Commit ~~a.txt~~ -m"modified".

o/p: Changes not staged for Commit:
modified: a.txt

So, first we should add modifications, then Commit.

$ git add a.txt

$ git Commit -m"modified".

<--- That's all --->

Golden Rule:-
only Commit related changes.
1 Commit = 1 Topic

Commit history:-

$ git log => gives the overview of history

$ git log --stat => To view more metadata

$ git log -p => individual changes

Ignoring files:-

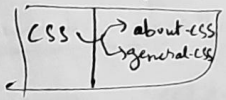
In project, Couple of below files will be there.

-> Temporary files, Cache files, build files, log files, password/files

-> But this could not be included in VCS.

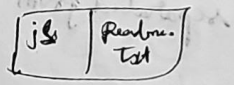
We need to tell git that we can exclude by using simple text file

Ignore one specific file, [path/to/file.ext]

Eg: 

, => CSS/general-CSS

Ignore all files with Certain Name [filename.ext]

Eg: 

Ignore all files of certain type [*.ext]

Ex: *.txt

Ignore all files in certain folder [path/to/folder/*]

Ex: js/*

Step1: need to create .gitignore file (can be done previously)

$ vi .gitignore → for all team members

Then add some files in that we exclude,

Ex: .DSStore

Save the file

$ git status → we can check, it will be excluded.

(*) Only untracked files can be ignored.

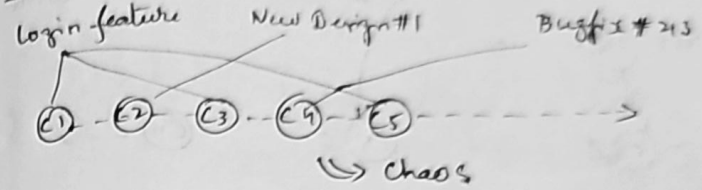
If we want to exclude in our only machine without affecting team member,

$ vi .git/info/exclude

10. Introduction to branches

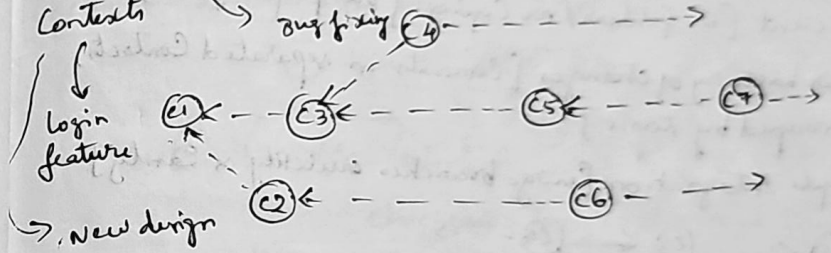
→ Branches keep separate topics cleanly separated from each other.

A world without branches :-

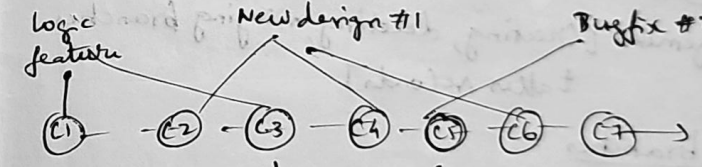


Branches keep separate topics separate

Each topic gets its own context completely isolated from all other contexts

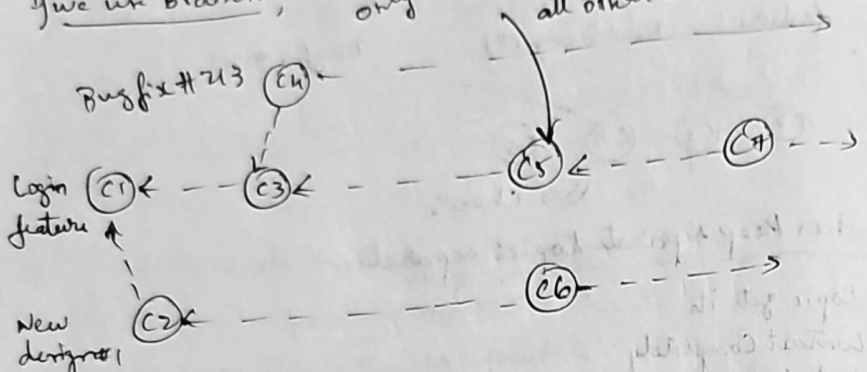


Why use branches?



Bug in "login feature" that affects all other topics, too

If we use branch, only "login feature" is affected by bug all other code continues to work.

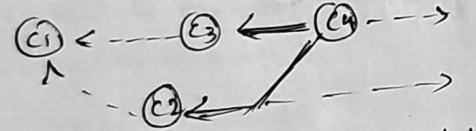


use Cases:

→ Efficient [No folders / no Copies of our project files]

→ Clear history of changes [Commits in separated Contexts, grouped by topic]

→ Simple integration [merge branches Quickly & Easily]

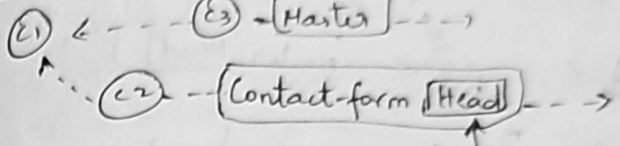


→ Easy Management [Creating, deleting, merging branches takes seconds]

When to use branches

→ A Branch for each New topic
New features, experiments, bug fixes...

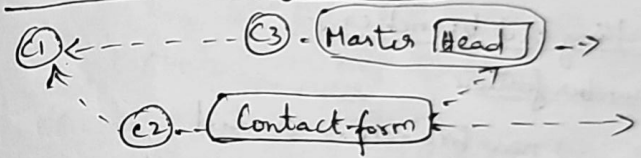
Head - The Currently Active Branch:



at any time, only one branch is active, that is called Checkout (or) Head branch

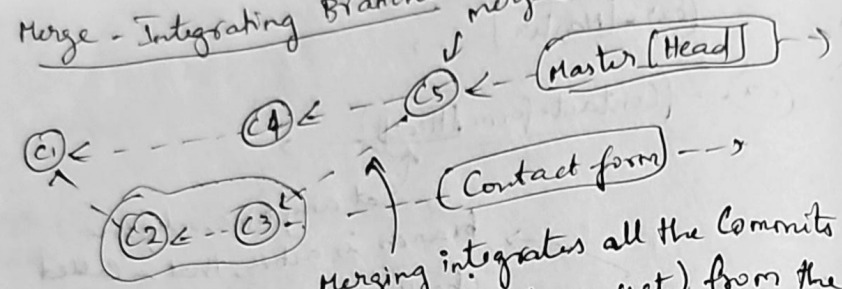
In Git, we are always working on a branch

Checkout - Switching the Head:



"Checkout" Command moves the HEAD pointer to a different branch and thereby makes that branch active

Merge - Integrating Branches



Merging integrates all the Commits (ones we don't have, yet) from the specified branch into your Current HEAD.

(11) creating & checking out branches :-

Eg: Going to add emp-number feature:-

$ git branch emp → new branch is created

$ git branch -v ⇒ to see all of our local branches.

o/p: emp d4cboab all done ← newly created branch
" " headbranch [* ⇒ active one]
* main "

$ vi e.txt → Contains emp-ID.

$ git status

o/p: on branch main *
untracked files
e.txt

$ git checkout emp → moving to emp ~~master~~ [active the emp]

o/p: Switched to branch 'emp'

$ git status

o/p: on branch emp
untracked files:-
e.txt

$ git add e.txt ⟶ added to staging area [only on emp branch]

$ git status

o/p: on branch emp
changes to be committed:
new file: e.txt

$ git Commit -m "emp is added"

o/p: [emp ...] emp is added
1 file changed, 1 insertion(+)
Create mode 100644 e.txt

$ git status

o/p: on branch emp
nothing to Commit, working tree clean

$ git branch -v

* emp e0d159b emp is added
main d4cboab all done

active

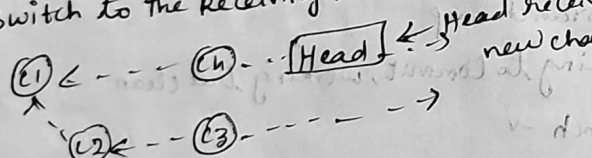
$ git checkout main
o/p switched to branch 'main'
$ git branch -v
emp e0d(sqb emp is added
*main d4cb0ab all done
$ ls
a.txt b.txt e.txt → where e.txt?

Now,
$ git checkout emp
$ ls
a.txt b.txt e.txt ← here, because we add this file to emp branch, not in master branch.

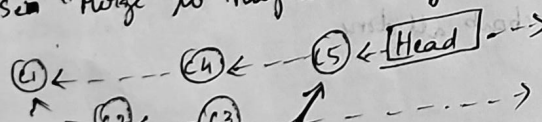
(12) Merging Branches :-

step 1 :-

(1) Switch to the Receiving Branch



(2) Use a "Merge" to Integrate changes



$ git branch -v
o/p * emp
main
$ git checkout main
$ git branch -v
o/p emp
* main
$ git merge emp
o/p updating d4cb0a..6
Fast-forward
e.txt | 1 +
1 file changed, 1 insertion(+)
$ ls
a.txt b.txt e.txt ← here 0 clues now, because we merged.

Very Important note

* We can't Create any branch until we Commit atleast one file to main/master branch.

(13) using stash :-

stash → Saving changes temporarily without Commit

why it is useful?

Golden rule; never Commit half done work.
- Even if it is small, it should be tested, realistic one.

Assume we are working in login feature, if bug comes. We need to do task in bug branch. so need to store temporary

$ git status
on branch master
changes not staged for commit:
modified a.txt →

Now if we are not completing work, we need to work on other branch & we also don't want to commit the unfinished work.

Stash → helps → save temporarily.

$ git stash
o/p: saved working directory and index state WIP on master: a3de794 Restructure page
Head is now at a3de794 Restructure pages

$ git status
on branch master
nothing to commit, working directory clean.

$ git stash list → To view the current stash
stash@{0}: WIP on master: a3de794 Restructure pages

↓ latest is always ↑
↓ saved one

Now we completed bug work [other branch], we want to resume the previous work where we left.

$ git stash apply stash@{0} ← our previous
o/p: Removing new_page.html
on branch master
changes not staged for commit:
modified: a.txt ← Now we resumed.

$ git stash ~~list~~ drop stash@{0} ← we removed stored one in stash list
o/p: Dropped stash@{0} (8881dd7 ...)

$ git stash list
o/p: No ⇔ empty, right? (:)

other method to ~~remove~~ stash list → resume our work in

$ git stash pop

when to use stash?

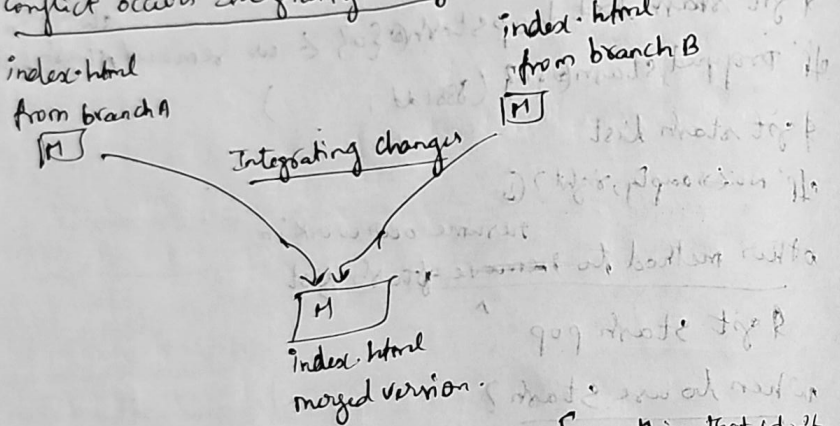
- before checking out different branch
- before pulling remote changes
- before merging (or) rebasing a branch

14. Dealing with Merge Conflicts

① undo & start fresh"
→ we can always undo a merge and start over.

② our problem only"
→ Conflicts don't bring our team to halt. They only affect us.
(i.e) Conflict will arise only in local developer machine only.

Conflict occurs Integrating changes:-



we can go our local editor → & change. [something that i don't understand in html]

[(i.e) Branch A & Branch B are there. They have moved one.]

Eg: $ git merge development → one of branch
of Conflict (content): Merge Conflict in index.html
Automatic merge failed; fix the Conflicts & then Commit the result.

(i.e) if two files are changed at same time, Some what git will figure out & do the merging But if two people change the same lines in the same file, then we have to solve the Conflict.

Git automatically shows the Conflict in "<<< " & ">>>"

Eg: <<<<<< (HEAD) ← branch
<h1> Welcome to our Git Crash Course! </h1>
=======
<h1> Welcome to this Git Crash Course! </h1>
>>>>>> (development) ← branch → these two branches only

changed to (in editor)
<h1> Welcome to Git best Crash Course! </h1>

undo the merge:-

(i.e) $ git merge --abort

(15) undoing things:-

$ ls
a.txt b.txt c.txt
$ git add a.txt b.txt ← two added.
$ git commit -m "wrong commit".
:
$ git status
:
(wrong commit) ← last commit shows here.

Now we have forget c file to add, need to do undo.

$ git add c.txt ← now we have added

$ git Commit --amend -m "This is the Correct Commit".

$ git status

This is the Correct Commit ← This replaces the previous Commit "wrong Commit".

one rule :-

Never amend the Commit that was already published in the remote server. If we do, it will rewrite that Commit, we will get trouble.

undoing local changes.

$ git status

on branch master

changes not staged for Commit:

modified : about.html

modified : index.html

$ git checkout HEAD index.html ⇒ here we are undoing the changes that we did for index.html

$ git status

on branch master

changes not staged for Commit:

modified : about.html ← No index.html file.

If we want to reset all changes,

$ git reset --hard HEAD ← discard all local changes.

HEAD is now at 3557773 Complete Commit

Revert the Commit :-

When we notice that changes are wrong (or) introducing bug, we need to revert the Commit.

Git doesn't delete any Commits, infact it creates new Commit, one that reverts of the other Commit, effectively undoing it.

$ git log

commit 8 ← latest Commit

: hello.txt

commit 48

index.html

:

commit (25) ← If we want to revert this Commit, let's go

change headlines

:

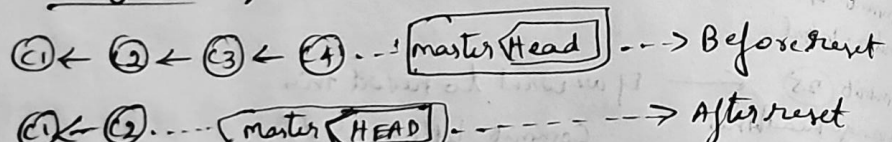
$ git show 25
:show] ← shows the changes in Commit
:

$ git revert (25) ← hash of that Commit
o/p: [master -] Revert "change headlines"
2 files changed...
$ git log
:commit 76
Revert "change headlines" ← Latest Commit
:commit 8
:
commit 48
:
commit 25 ← It is also here.
change headlines
:

(i.e) Reverted

Different Way to undo Commit:-

Resetting HEAD / also called Rolling back



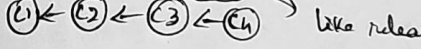
After reset, C3 & C4 are no longer visible in this branch

$ git log
Commit 2 ←
:
Commit 45
:
commit 27
Index.html
:
commit 18
:

let's rollback to 27,
$ git ~~revert~~ reset --hard (27)
$ git log
commit 27
index.html
commit 18
] => Here 2 & 45 are gone.

(16) Tags:-

↳ allowed to mark important points in project's history.



like release info.

$ git checkout empid
$ ... do one Commit ["Commit - Index.html"]
$ git tag → list tags that we created
$ No output => Because we didn't create yet.

$ git tag -a 2.0 -m "version 2.0 with high logic screen"

We → annotated tag (i.e.) ← adding more info

$ git show 2.0

o/p:

Author - - -

Date - - -

version 2.0 with high logic screen

:

Commit 27

Index.html

] ← This will only come. Because tag is added to that latest commit only.

We can add tag to any commit,

$ git tag -a 3.0 45 -m "version 3.0"

45 → Commit ID

That's all.

Deleting tag

$ git tag

1.0

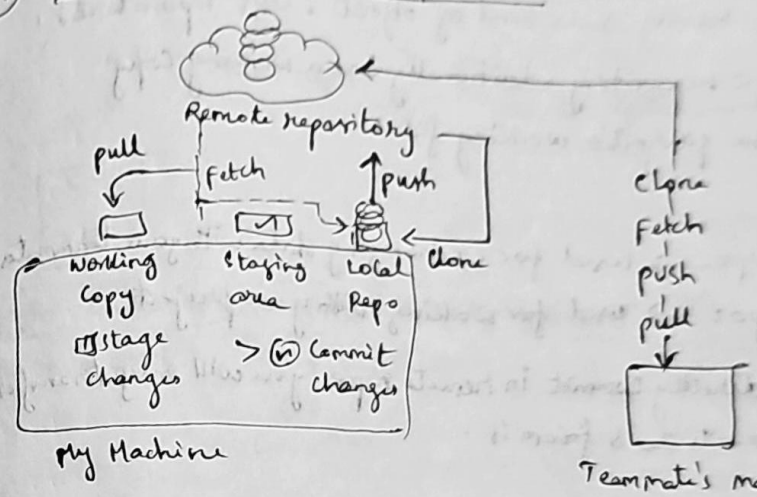
2.0

3.0

$ git tag -d 3.0

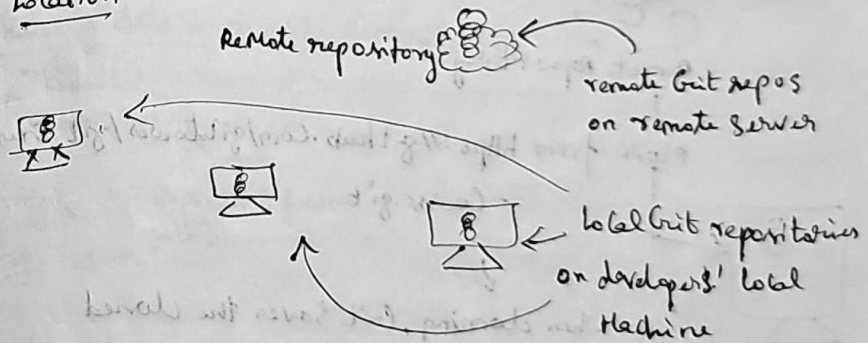
Deleted tag '3.0' [redacted].

(17) Introduction to remote Repositories



Difference between Local & Remote Repos

① Location



② Features :-

→ Both are having same kind of object : Git repositories !

→ Local git repository additionally has a working copy with the project's working files.

③ Usage :-

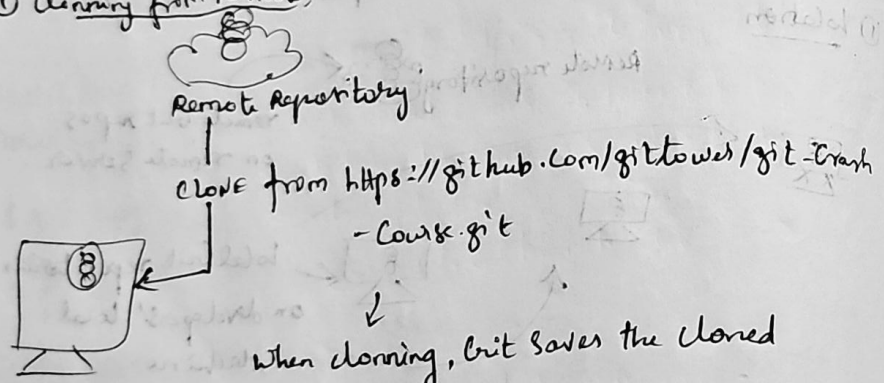
Remote repo : → used for exchanging data with your teammates.

Local repo : → used for working with your project

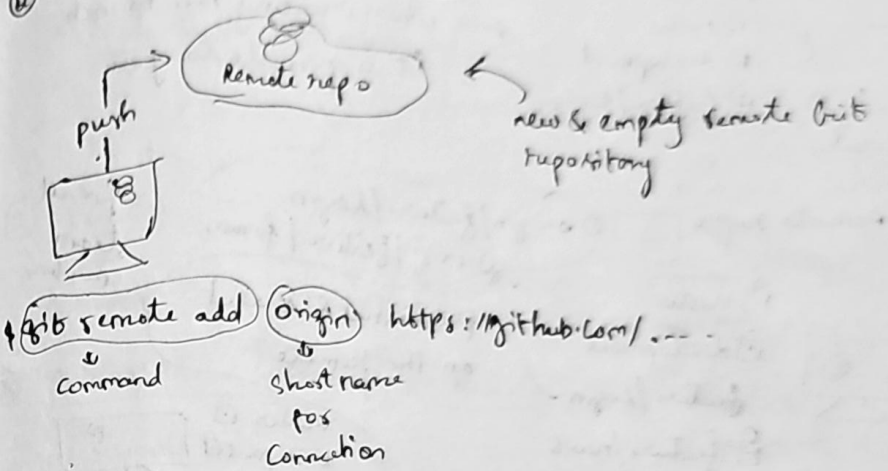
you never directly commit in remote repo, you will only transfer existing commits to & from it.

⑱ Connecting a Remote Repository :-

① Cloning from Remote :-



② Adding a Remote Connection :-

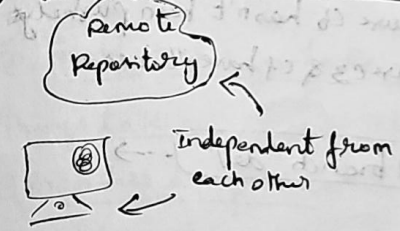


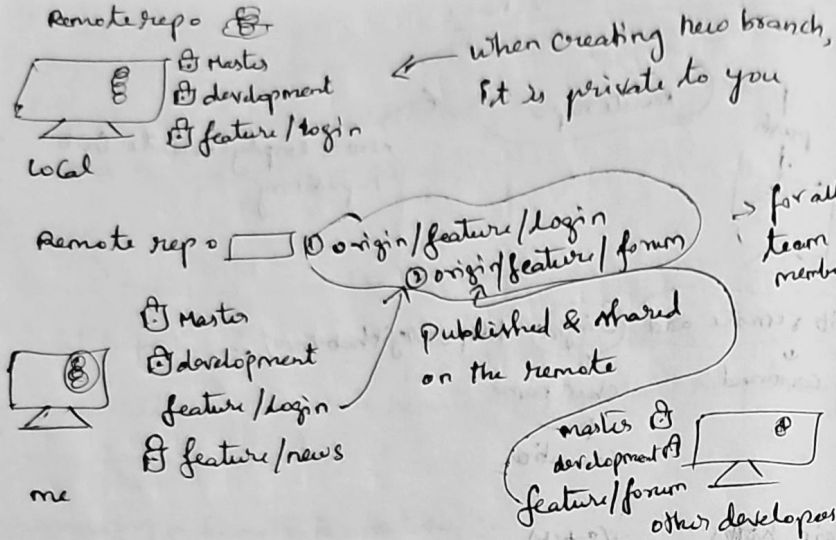
or

origin https ... (fetch)

origin https ... (push)

⑲ sharing data on Remote Repository :-

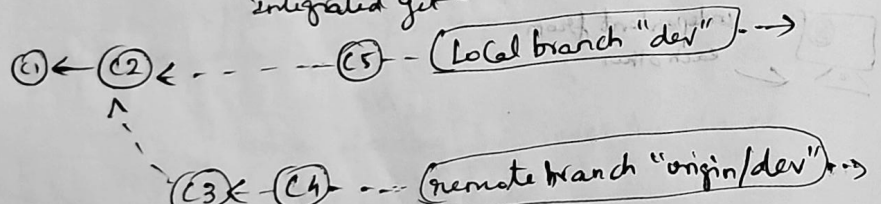




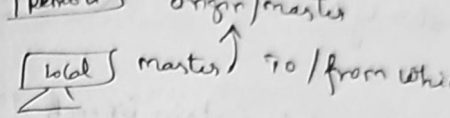
Tracking Connections

Local "dev" tracks remote "origin/dev" and is

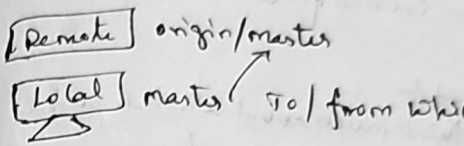
1. Commit ahead because C5 hasn't been pushed
2. Commits behind because C3 & C4 haven't been integrated yet



① To which?



②



⑳ Publishing a Local Repo on Remote

$ git remote add origin https...

$ git push -u origin master

origin → remote repo Connection

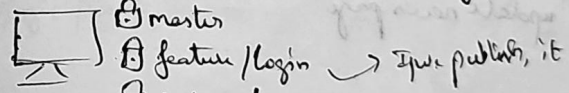
master → which remote branch

㉑ pushing changes to Remote

Things to Know:-

① Branches are private

Local branches are private by default



② Pushing/Pulling Concerns Branches

[We can always push/pull only branches not individual Commits]

(*) push/pulling for all new Commits for certain branch.

③ HEAD is pushed

when we push, we don't need to think much about what is pushed. we always push the current local HEAD branch.

```
$ git checkout feature/news
Switched to branch 'feature/news'
$ git push -u origin feature/news
```

Qs:

```
$ git branch -vva   → for also see tracking connection.
* feature/news  85225d7   first version of 'news' page
  master        a3de794
  remote/origin/feature/news  85225d7
$ git add -A  [just add anything]
$ git commit -m "update news page"
```

```
$ git status
On branch feature/news
Your branch is ahead of 'origin/feature/news' by 1 commit
```

(tracking connection see diagram for understanding)

so, now from above o/p, it reminds us [we are ahead], so we can push

```
$ git push -u   [Here we don't need to tell which remote & branch, bcoz from status we understood]
```

Qs:--

another special situation:-

If your teammate also pushes to this branch, this might be your push fail.

This because Git warned you when there are changes in the Remote that you haven't integrated yet.

```
$ git push
To https://git ....
[rejected] feature/news → feature/news (fetch first)
error: failed to push some refs to 'https://github.com/gittavey'
```

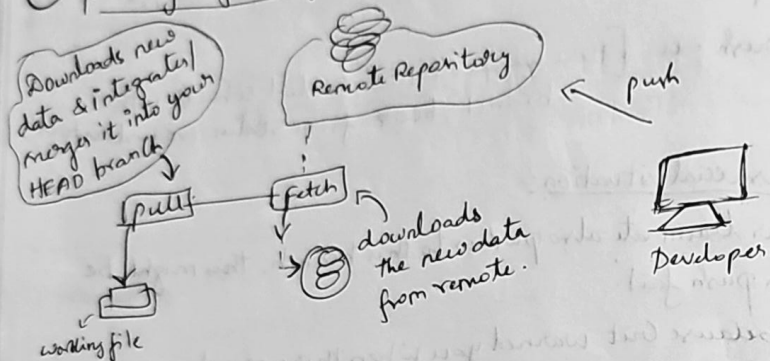
so first need to integrate,

```
$ git pull
```

now $ git push

It is pushed fine

(22) Pulling & fetching changes from Remote:-



$ git branch -vva

* master a3de794 [origin/master] Restr. pages
remotes/origin/master a3de794 Restr. pages

$ git fetch origin ← fetching

remote: Counting objects: 5, done.
remote: Compressing objects: 100%.

a3de794 .. master → origin/master
* [new branch] feature/signup → origin/feature/signup

Here we can see new change is there.

$ git branch -vva

master a3de794 [origin/master: behind 1] Restr. p
remotes/origin/feature/signup 0fd6efe Add Login interface
remotes/origin/master 01738fo New headline for index page

So our master branch is behind, so pull.

$ git pull

updating a3de794.

$ git branch -vva

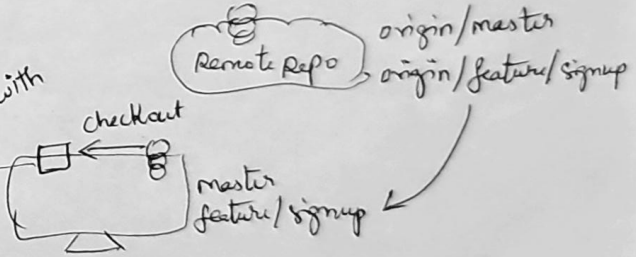
* master 01738fo [origin/master] New headline

Now master is updated

That's all

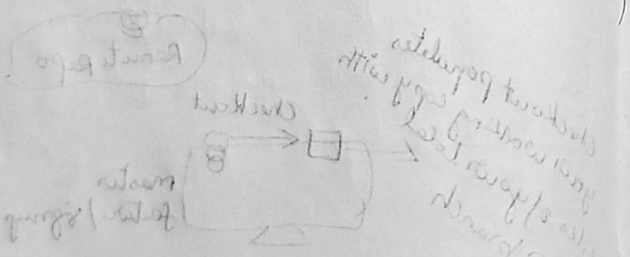
Now we can see new branch remote/origin/feature/signup in remote, Can we work in that branch? Let's go.

But wait, it is in remote, we can't work on remote.



$ git checkout --track origin/feature/signup (from origin)
Branch feature/signup set up to track remote branch feature/signup from origin.
Switched to new branch feature/signup

$ git branch -vva → we pulled & started [copied] to our local repo
* feature/signup ofd6ef1 [origin/feature/signup] Add login
master 01738fo
remotes/origin/feature/signup ofd6ef1 Add login into fac
remotes/origin/master 01738fo New headline for index



→ Renesas RCar M3, Qualcomm SA8195
→ XM/3DARS (x38f), STA710
→ RSU
→ DOORS