**Experiment No 1**

**Aim:** To implement basic concepts of CPP.

**Theory:**

**Introduction :**

C++ is a cross-platform language that can be used to create high-performance applications.C++ was developed by Bjarne Stroustrup, as an extension to the [C language](https://www.w3schools.com/c/index.php).C++ gives programmers a high level of control over system resources and memory.

## Difference between C and C++:

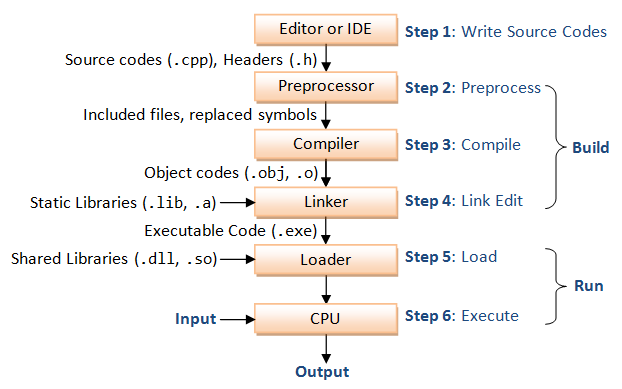
C++ was developed as an extension of [C](https://www.w3schools.com/c/index.php), and both languages have almost the same syntax. The main difference between C and C++ is that C++ support classes and objects, while C does not.

## Why Use C++:

C++ is one of the world's most popular programming languages. C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems. C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs. C++ is portable and can be used to develop applications that can be adapted to multiple platforms.

C++ is fun and easy to learn! As C++ is close to [C](https://www.w3schools.com/c/index.php), [C#](https://www.w3schools.com/cs/index.php) and [Java](https://www.w3schools.com/java/default.asp), it makes it easy for programmers to switch to C++ or vice versa.

**Writing and Executing a C++ Program:**



**Output Statement(cout):**

The cout stream is used in conjunction with the overloaded operator << (a pair of “less than” signs).

Syntax:

cout<<variable1<<variable2<<…………………….<<variablen;

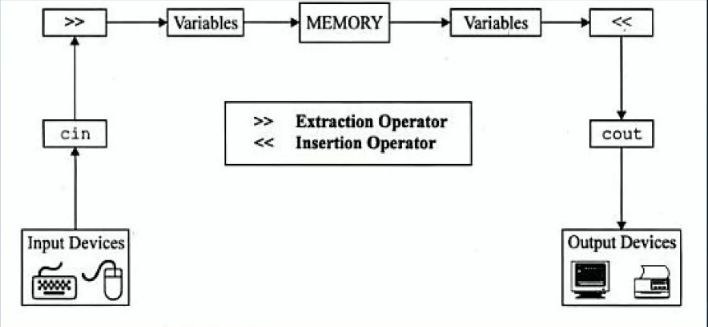
The << operator is known as insertion operator since it inserts the data that follows it into the stream that precedes it.

**Input Statement(cin):**

Handling the standard input in C++ is done by applying the overloaded operator >> (a pair of greater than sign) known as extraction operator on the cin stream. This must be followed by the variable that will store the data that would by keyed in.

Syntax:

Cin>>variable1>>variable2>>…………………….>>variablen;

****

## C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

* int - stores integers (whole numbers), without decimals, such as 123 or -123
* double - stores floating point numbers, with decimals, such as 19.99 or -19.99
* char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* string - stores text, such as "Hello World". String values are surrounded by double quotes
* bool - stores values with two states: true or false

### Syntax

typevariableName*=*value*;*

typevariableName*;*

typevariable1>>variable2>>…………………….>>variablen;

The general rules for naming variables are:

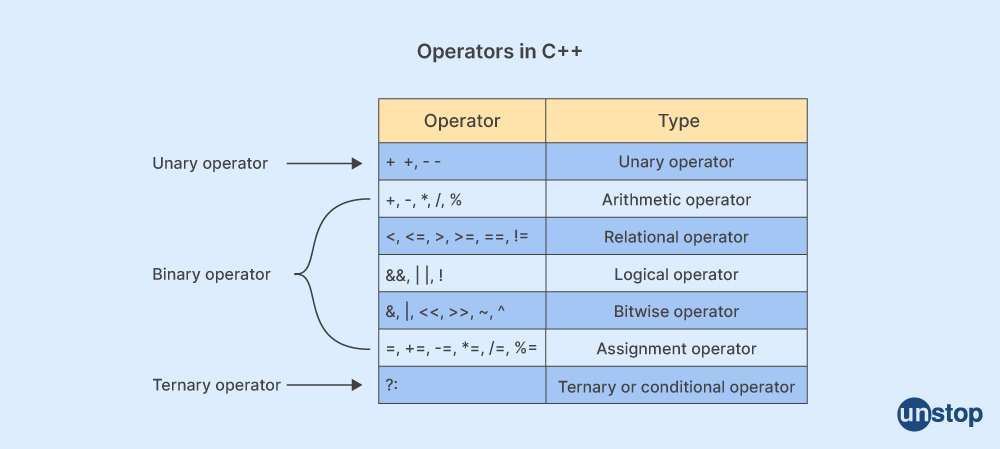
* Names can contain letters, digits and underscores
* Names must begin with a letter or an underscore (\_)
* Names are case-sensitive (myVar and myvar are different variables)
* Names cannot contain whitespaces or special characters like !, #, %, etc.
* Reserved words (like C++ keywords, such as int) cannot be used as names

**Data Types:**



**Operators in C++ can be classified into 6 types:**

1. Arithmetic Operators
2. Relational Operators
3. Logical Operators
4. Bitwise Operators
5. Assignment Operators
6. Ternary or Conditional Operators



**Practical Related Questions:**

1.Write the advantages and disadvantages of C++.

2. Give the structure of C++ program.

3. List 5 rules for forming variable names.

4. Define:

a. identifiers

b. keywords

c. constants

5. Which are the input and output operator in C++? Give its example.

6. Write a note on Scope of Variables.

**Programs :**

1. CPP Program to design a simple calculator using **Arithmetic Operators**
2. CPP Program to find whether the entered number is greater or smaller.
3. CPP Program to find the smaller of three numbers using **Logical Operators**
4. CPP Program to swap two numbers using **Assignment Operators**
5. CPP Program to solve the following expression **(3\*i-2\*j)%(2\*a-b)**

**Conclusion :**

Hence, we learnt to implement the basic concepts of CPP.

**PRACTICAL 1**

**WAP in cpp program to design a simple calculator using arithmetic operator**

#include <iostream.h>

#include<conio.h>

int main()

{

float num1,num2,add,mul,sub,div;

cout << "Enter first number: ";

cin >> num1;

cout << "Enter second number: ";

cin >> num2;

add=num1+num2;

sub=num1-num2;

mul=num1\*num2;

div=num1/num2;

cout << "Addition = " << add << endl;

cout << "Subtraction = " << sub << endl;

cout << "Multiplication = " << mul << endl;

cout << "Division = " << div << endl;

return 0;

}

**WAP in CPP to find whether the entered number is greater or smaller**

#include <iostream.h>

#include<conio.h>

int main()

{

int num1, num2;

cout << "Enter the first number: ";

cin >> num1;

cout << "Enter the second number: ";

cin >> num2;

if (num1 > num2)

{

cout << num1 << " is greater than " << num2 << endl;

}

else if (num1 < num2)

{

cout << num2 << " is greater than " << num1 << endl;

}

else

{

cout << "Both numbers are equal." << endl;

}

return 0;

}

**WAP in CPP to find the smaller of three numbers using Logical Operators**

#include <iostream.h>

#include<conio.h>

int main()

{

int num1, num2, num3;

cout << "Enter three numbers: ";

cin >> num1 >> num2 >> num3;

if (num2 < num1 && num2 < num3) {

cout << "The smallest number is: " << num2 << endl;

} else if (num3 < num1 && num3 < num2) {

cout << "The smallest number is: " << num3 << endl;

}

else

{

cout << "The smallest number is: " << num1 << endl;

}

return 0;

}

**WAP in CPP to swap two numbers using Assignment Operators**

#include <iostream.h>

#include<conio.h>

int main()

{

int a, b;

cout << "Enter two numbers: ";

cin >> a >> b;

// Swapping using assignment operators

a = a + b;

b = a - b;

a = a - b;

cout << "After swapping: a = " << a << ", b = " << b << endl;

return 0;

}

**OR**

#include <iostream.h>

#include<conio.h>

int main()

{

int a, b,t;

cout << "Enter value of a: ";

cin >> a;

cout << "Enter value of b: ";

cin >> b;

// Swapping using assignment operators

t=a;

a=b;

b=t;

cout << "After swapping: a = " << a << ", b = " << b << endl;

return 0;

}

**WAP in CPP to solve the following expression (3\*i-2\*j)%(2\*a-b)**

#include <iostream.h>

#include<conio.h>

int main()

{

int i, j, a, b, result;

cout << "Enter the values of i, j, a, and b: ";

cin >> i >> j >> a >> b;

// Calculate the numerator and denominator

int numerator = 3 \* i - 2 \* j;

int denominator = 2 \* a - b;

// Check for division by zero

if (denominator == 0)

{

cout << "Error: Division by zero" << endl;

} else

{

// Perform the modulo operation

result = numerator % denominator;

cout << "Result: " << result << endl;

}

return 0;

}

**Experiment No 2**

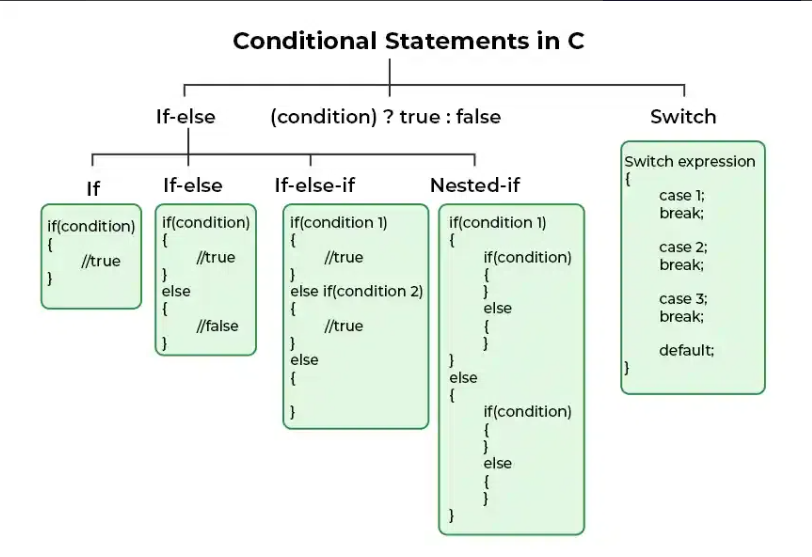
**Aim:** To implement the control statements in CPP.

**Theory:**

The **conditional statements** (also known as decision control structures) such as if, if else, switch, etc. are used for decision-making purposes in C++ programs.They are also known as Decision-Making Statements and are used to evaluate one or more conditions and make the decision whether to execute a set of statements or not. These decision-making statements in programming languages decide the direction of the flow of program execution.

## Need of Conditional Statements

There come situations in real life when we need to make some decisions and based on these decisions, we decide what should we do next. Similar situations arise in programming also where we need to make some decisions and based on these decisions we will execute the next block of code. For example, in C if x occurs then execute y else execute z. There can also be multiple conditions like in C if x occurs then execute p, else if condition y occurs execute q, else execute r. This condition of C else-if is one of the many ways of importing multiple conditions.

****

**if statement:**

The if in C++ is a decision-making statement that is used to execute a block of code based on the value of the given expression. It is one of the core concepts of C++ programming and is used to include conditional code in our program.

## Syntax of if Statement

if(condition)

{

// if body

// Statements to execute if condition is true

}

**if-else statement:**

The if-else statement is a decision-making statement that is used to decide whether the part of the code will be executed or not based on the **specified condition (test expression)**. If the given condition is true, then the code inside the if block is executed, otherwise the code inside the else block is executed.

## Syntax of if-else

if (condition)

{

// code executed when the condition is true

}

else

{

// code executed when the condition is false

}

# if else if ladder

**The if else if** ladder in C++ programming is used to test a series of conditions sequentially. Furthermore, if a condition is tested only when all previous if conditions in the if-else ladder are false. If any of the conditional expressions evaluate to be true, the appropriate code block will be executed, and the entire if-else ladder will be terminated.

# ****Syntax of**** if else if ladder**:**

// any if-else ladder starts with an if statement only

if(condition)

{

……………..

}

else if(condition)

{

// this else if will be executed when condition in if is false and

// the condition of this else if is true

}

.... // once if-else ladder can have multiple else if

else

{ // at the end we put else

………..........

}

# Switch Statement:

Switch case statement evaluates a given expression and based on the evaluated value(matching a certain condition), it executes the statements associated with it. Basically, it is used to perform different actions based on different conditions(cases).

* Switch case statements follow a selection-control mechanism and allow a value to change control of execution.
* They are a substitute for long [if statements](https://www.geeksforgeeks.org/decision-making-c-c-else-nested-else/) that compare a variable to several integral values.
* The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

## Syntax of switch Statement

**switch(expression)**

**{**

**case** value1**:** statement\_1;

**break;**

**case** value2**:** statement\_2;

**break;**

.

.

.

**case** value\_n**:** statement\_n;

**break;**

**default:** default\_statement;

}

**Practical Related Questions:**

1. Explain break statement with its syntax
2. Explain continue statements with its syntax.
3. Write a note on goto statement.
4. Draw a flow chart for switch case.

**Programs :**

1. Write a program that prompts the user for a number. If the number is even, print "The number is even. " Otherwise, print "The number is odd. **(if else statement)**
2. Design a program that simulates a simple grade calculator. Ask the user for their exam score (out of 100). Based on the score, print the corresponding letter grade:

* A: 90-100
* B: 80-89
* C: 70-79
* D: 60-69
* F: Below 60 **(if-else ladder)**

1. Design a program that acts as a simple menu driven program to find square, square root, cube , perform the corresponding calculation and exit the program using **switch case.**

**Conclusion :**

Hence, we learnt to implement the conditional statements of CPP.

**PRACTICAL 2**

**Write a program that prompts the user for a number. If the number is even, print "The number is even. " Otherwise, print "The number is odd.**

#include <iostream.h>

#include<conio.h>

int main()

{

int num;

cout << "Enter a number: ";

cin >> num;

if (num % 2 == 0)

{

cout << "The number is even." << endl;

}

else

{

cout << "The number is odd." << endl;

}

return 0;

}

**Design a program in cpp that simulates a simple grade calculator. Ask the user for their exam score (out of 100) of 5 subjects .calculate the total and averade Based on the average, print the corresponding letter grade:**

**1.     A: 90-100**

**2.     B: 80-89**

**3.     C: 70-79**

**4.     D: 60-69**

**5.     F: Below 60**

#include <iostream.h>

#include<conio.h>

int main()

{

int subject1, subject2, subject3, subject4, subject5;

float total, average;

cout << "Enter marks for 1st subjects (out of 100): " << endl;

cin >>subject1;

cout << "Enter marks for 2nd subjects (out of 100): " << endl;

cin >>subject2;

cout << "Enter marks for 3rd subjects (out of 100): " << endl;

cin >>subject3;

cout << "Enter marks for 4th subjects (out of 100): " << endl;

cin >>subject4;

cout << "Enter marks for 5th subjects (out of 100): " << endl;

cin >>subject5;

total = subject1 + subject2 + subject3 + subject4 + subject5;

average = total / 5;

cout << "Total marks: " << total << endl;

cout << "Average marks: " << average << endl;

if (average >= 90)

{

cout << "Grade: A" << endl;

}

else if (average >= 80)

{

cout << "Grade: B" << endl;

}

else if (average >= 70)

{

cout << "Grade: C" << endl;

}

else if (average >= 60)

{

cout << "Grade: D" << endl;

}

else

{

cout << "Grade: F" << endl;

}

return 0;

}

**Design a program that acts as a simple menu driven program to find square, square root, cube , perform the corresponding calculation and exit the program using switch case.**

#include <iostream.h>

#include <cmath>

#include<conio.h>

int main()

{

int choice, num;

double result;

do

{

cout << "\nMenu:\n";

cout << "1. Find square\n";

cout << "2. Find square root\n";

cout << "3. Find cube\n";

cout << "4. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter a number: ";

cin >> num;

result = num \* num;

cout << "Square of " << num << " is: " << result << endl;

break;

case 2:

cout << "Enter a number: ";

cin >> num;

if (num >= 0)

{

result = sqrt(num);

cout << "Square root of " << num << " is: " << result << endl;

}

else

{

cout << "Square root of negative number is not defined." ;

}

break;

case 3:

cout << "Enter a number: ";

cin >> num;

result = num \* num \* num;

cout << "Cube of " << num << " is: " << result << endl;

break;

case 4:

cout << "Exiting program..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 4);

return 0;

}

**Experiment No 3**

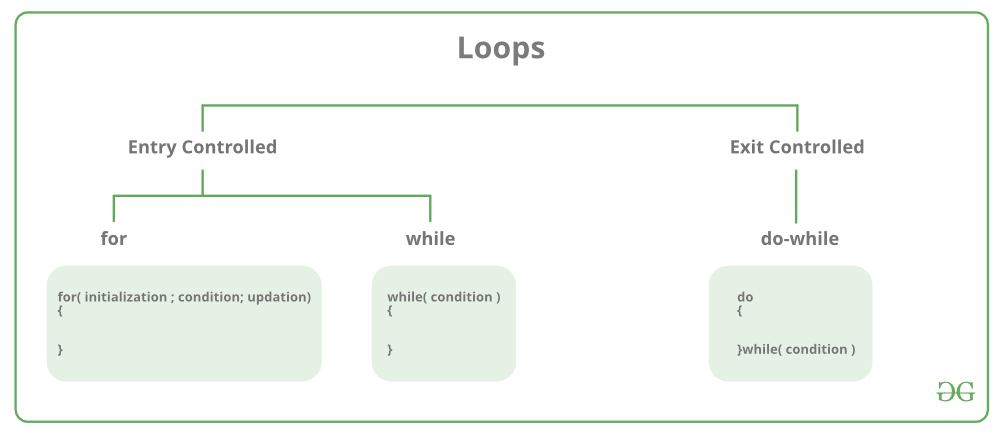
**Aim:** To implement the control statements in CPP.

**Theory:**

In Programming, sometimes there is a need to perform some operation**more than once** or (say) **n number**of times. Loops come into use when we need to repeatedly execute a block of statements.

## ****There are mainly two types of loops:****

1. **Entry Controlled loops**: In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loop** is entry-controlled loops.
2. **Exit Controlled Loops**: In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the do-while**loop** is exit controlled loop.



## For Loop-

A *For loop* is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

**Syntax:**

**for (initialization expr; test expr; update expr)**  
{   
 // body of the loop  
 // statements we want to execute  
}

### Explanation of the Syntax:

* **Initialization statement:** This statement gets executed only once, at the beginning of the for loop. You can enter a declaration of multiple variables of one type, such as int x=0, a=1, b=2. These variables are only valid in the scope of the loop. Variable defined before the loop with the same name are hidden during execution of the loop.
* **Condition:** This statement gets evaluated ahead of each execution of the loop body, and abort the execution if the given condition get false.
* **Iteration execution:** This statement gets executed after the loop body, ahead of the next condition evaluated, unless the for loop is aborted in the body (by break, goto, return or an exception being thrown.)

### NOTES:

* The initialization and increment statements can perform operations unrelated to the condition statement, or nothing at all – if you wish to do. But the good practice is to only perform operations directly relevant to the loop.
* A variable declared in the initialization statement is visible only inside the scope of the for loop and will be released out of the loop.
* Don’t forget that the variable which was declared in the initialization statement can be  modified during the loop, as well as the variable checked in the condition.

IMG_256

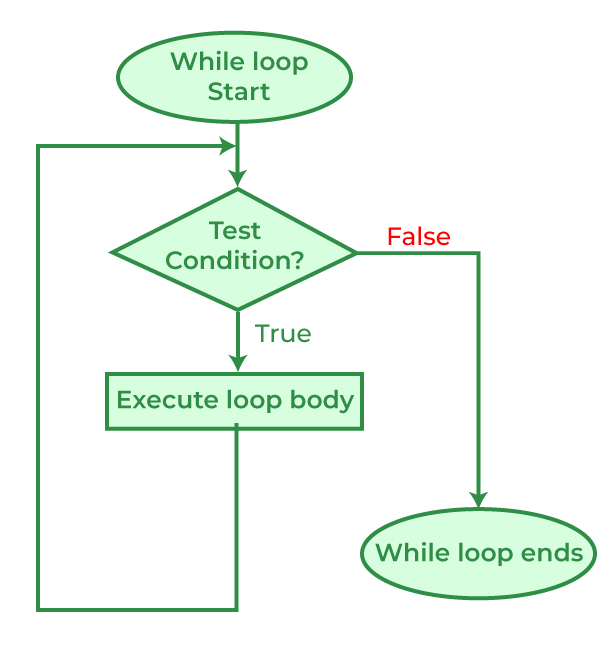
## While Loop-

While studying **for loop** we have seen that the number of iterations is ***known beforehand***, i.e. the number of times the loop body is needed to be executed is known to us. while loops are used in situations where **we do not know** the exact number of iterations of the loop **beforehand**. The loop execution is terminated on the basis of the test conditions.  
We have already stated that a loop mainly consists of three statements – initialization expression, test expression, and update expression. The syntax of the three loops – For, while, and do while mainly differs in the placement of these three statements.

**Syntax**:

**initialization expression;**  
while (**test\_expression**)  
{  
 // statements  
   
 **update\_expression;**  
}

### ****Flow Diagram of while loop:****



## Do-while loop

In Do-while loops also the loop execution is terminated on the basis of test conditions. The main difference between a do-while loop and the while loop is in the do-while loop the condition is tested at the end of the loop body, i.e do-while loop is exit controlled whereas the other two loops are entry-controlled loops.   
**Note**: In a do-while loop, the loop body will ***execute at least once*** irrespective of the test condition.  
**Syntax**:

**initialization expression;**  
do  
{  
 // statements  
 **update\_expression;**  
} while (**test\_expression**);

***Note****: Notice the semi – colon(“;”)in the end of loop.*

### ****Flow Diagram of the do-while loop:****

### IMG_256

### ****Advantages :****

1. **High performance**: C++ is a compiled language that can produce efficient and high-performance code. It allows low-level memory manipulation and direct access to system resources, making it ideal for applications that require high performance, such as game development, operating systems, and scientific computing.
2. **Object-oriented programming:** C++ supports object-oriented programming, allowing developers to write modular, reusable, and maintainable code. It provides features such as inheritance, polymorphism, encapsulation, and abstraction that make code easier to understand and modify.
3. **Wide range of applications:** C++ is a versatile language that can be used for a wide range of applications, including desktop applications, games, mobile apps, embedded systems, and web development. It is also used extensively in the development of operating systems, system software, and device drivers.
4. **Standardized language:** C++ is a standardized language, with a specification maintained by the ISO (International Organization for Standardization). This ensures that C++ code written on one platform can be easily ported to another platform, making it a popular choice for cross-platform development.
5. **Large community and resources:** C++ has a large and active community of developers and users, with many resources available online, including documentation, tutorials, libraries, and frameworks. This makes it easy to find help and support when needed.
6. **Interoperability with other languages:** C++ can be easily integrated with other programming languages, such as C, Python, and Java, allowing developers to leverage the strengths of different languages in their applications.

Overall, C++ is a powerful and flexible language that offers many advantages for developers who need to create high-performance, reliable, and scalable applications.

**Practical Related Questions:**

**Programs :**

**Write a program in CPP**

1. Palindrome
2. Fibonacci Series
3. Armstrong number

**Conclusion :**

Hence, we learnt to implement the looping statements of CPP.

**PRACTICAL 3**

**Write a Program in CPP to find whether the entered number is a palindrome or not.**

**Using For Loop**

#include <iostream>

using namespace std;

int main()

{

int i,n,rev,rem,original;

cout << "Enter an integer: ";

cin >> n;

original = n;

// Reverse the integer

for (i=n; i> 0; i/= 10)

{

rem = i % 10;

rev = rev \* 10 + rem;

}

cout << rev<<endl;

// Check if the original and reversed numbers are equal

if (original == rev)

cout <<"\n"<< n << " is a palindrome.";

else

cout << n << " is not a palindrome.";

return 0;

}

**Using While Loop**

#include <iostream>

using namespace std;

int main()

{

int n, reversed = 0, remainder, original;

cout << "Enter an integer: ";

cin >> n;

original = n;

// Reverse the integer

while (n != 0)

{

remainder = n % 10;

reversed = reversed \* 10 + remainder;

n /= 10;

}

// Check if the original and reversed numbers are equal

if (original == reversed)

cout << n << " is a palindrome.";

else

cout << n << " is not a palindrome.";

return 0;

}

**Using Do-While Loop**

#include <iostream>

using namespace std;

int main()

{

int n, reversed = 0, remainder, original;

cout << "Enter an integer: ";

cin >> n;

original = n;

// Reverse the integer

do

{

remainder = n % 10;

reversed = reversed \* 10 + remainder;

n /= 10;

} while (n != 0);

// Check if the original and reversed numbers are equal

if (original == reversed)

cout << n << " is a palindrome.";

else

cout << n << " is not a palindrome.";

return 0;

}

**Write a Program in CPP to print the Fibonacci series.**

**Using a For Loop**

#include <iostream>

using namespace std;

int main()

{

int n, t1 = 0, t2 = 1, nextTerm;

cout << "Enter the number of terms: ";

cin >> n;

cout << "Fibonacci Series: ";

for (int i = 1; i<= n; ++i)

{

cout << t1 << ", ";

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

}

return 0;

}

**Using a While Loop**

#include <iostream>

using namespace std;

int main()

{

int n, t1 = 0, t2 = 1, nextTerm;

cout << "Enter the number of terms: ";

cin >> n;

cout << "Fibonacci Series:";

int i = 1;

while (i <= n)

{

cout << t1 << ", ";

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

i++;

}

return 0;

}

**Using a Do-While Loop**

#include <iostream>

using namespace std;

int main()

{

int n, t1 = 0, t2 = 1, nextTerm;

cout << "Enter the number of terms: ";

cin >> n;

cout << "Fibonacci Series:";

int i = 1;

do

{

cout << t1 << ", ";

nextTerm = t1 + t2;

t1 = t2;

t2 = nextTerm;

i++;

} while (i <= n);

return 0;

}

**Write a Program in CPP to calculate Armstrong number of a entered number.**

**Using a For Loop**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int num, originalNum, remainder, n = 0, sum = 0;

cout << "Enter an integer: ";

cin >> num;

originalNum = num;

// Find the number of digits

while (originalNum != 0)

{

originalNum /= 10;

++n;

}

originalNum = num;

// Calculate sum of power of digits

for (int i = 0; i < n; ++i)

{

remainder = originalNum % 10;

sum += pow(remainder, n);

originalNum /= 10;

}

if (sum == num)

cout << num << " is an Armstrong number";

else

cout << num << " is not an Armstrong number";

return 0;

}

**Using a While Loop**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int num, originalNum, remainder, n = 0, sum = 0;

cout << "Enter an integer: ";

cin >> num;

originalNum = num;

// Find the number of digits

while (originalNum != 0)

{

originalNum /= 10;

++n;

}

originalNum = num;

// Calculate sum of power of digits

while (originalNum != 0)

{

remainder = originalNum% 10;

sum += pow(remainder, n);

originalNum /= 10;

}

if (sum == num)

cout << num << " is an Armstrong number";

else

cout << num << " is not an Armstrong number";

return 0;

}

**Using a Do-While Loop**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

int num, originalNum, remainder, n = 0, sum = 0;

cout << "Enter an integer: ";

cin >> num;

originalNum = num;

// Find the number of digits

do

{

originalNum /= 10;

++n;

} while (originalNum != 0);

originalNum = num;

// Calculate sum of power of digits

do

{

remainder = originalNum % 10;

sum += pow(remainder, n);

originalNum /= 10;

} while (originalNum != 0);

if (sum == num)

cout << num << " is an Armstrong number";

else

cout << num << " is not an Armstrong number";

return 0;

}

**Experiment No 4**

**Aim:** To implement basic concepts of Class and objects in CPP.

**Theory:**

**Specifying a class:**

A class is a way to bind the data and its associated functions together. It allows the data and functions to be hidden from the external use. Generally a class specification has two parts:

1. Class declaration
2. Class function Definition

The class declaration describes the data type and scope of its members. The class function describes how the class functions are implemented. The general form of the class declaration is:

*class class\_name*

*{*

*private:*

*variable declarations;*

*function declarations;*

*public:*

*variable declarations;*

*function declarations;*

*}*

The keyword class specifies, that what follows is an abstract data of type class\_name; the body of a class is enclosed within the braces and terminated by a semi-colon; the class body contains the declaration of the variables and functions, these variables and the functions are called as the class members. They are usually grouped under two sections namely, private and public to denote which of the members are private and which of them are public.The keywords private and public are known as the visibility labels. The variables declared inside the class are called as data members and the functions declared inside the class are called as the member functions.

**Creating Objects:**

Once the class has been declared, we can create variables of that type by using the class name. in CPP the class variables are known as objects. We can also create more than one objects in one statement. The declaration of an object is similar to that of the variable of any basic type. The necessary memory space is allocated to an object at this stage.

**Accessing Class Members:**

Following format is used for calling a member function.

*Object\_name.function\_name(actual\_arguments);*

**Defining Member Functions :**

Member functions can be defined in two places :

1. Outside the Class Definition
2. Inside the Class Definition

Irrespective of the place of definition, the function should perform the same task. The code for the function body would be identical in both the cases.

**Outside the Class Definition:**

The general form of a member function definition is:

*return\_type class\_name::function\_name(argument declaration)*

*{*

*function body*

*}*

The symbol : : is called as the scope resolution operator, it tells the compiler that the function function\_name belongs to the class class\_name. That is, the scope of the function is restricted to the class\_name specified in the header file. The member functions have some special characteristics that are often used in the program development. These characteristics are:

* Several different classes can use the same function name.
* Member functions can access the private date of the class. A non-member function do so. (An exception to this rule is a friend function.)
* A member function directly, without using the dot operator.

**Inside the Class Definition:**

Another method of defining a member function is to replace the function declaration by the actual function definition inside the class.

*class item*

*{*

*int number;*

*float cost;*

*public:*

*void getdata(int a,float b)*

*{*

*number=a,cost=b;*

*}*

*void putdata(void)*

*{*

*cout<<number;*

*cout<<cost;*

*}*

*};*

When the function is defined inside a class, it is treated as an inline function. Only small functions are defined inside the class definition.

**Memory Allocation for Objects:**

The memory space for objects is allocated when they are declared and not when the class is specified. Actually , the member functions are created and placed in the memory space only one when they are defined as a part of a class specification. All the objects belonging to that class use the same member functions, no separate space is allocated for member functions when the objects are created. On;y spaece for member variables is allocated separately for each object. Separate memory locations for the objects are essential, because the member variables will hold different data values for different objects.

**Access Specifiers:**

Data hiding is one of the important features of object oriented programming which allows preventing the functions of a programming which allows preventing the functions of a program to access directly the internal representation of a class type. Access specifiers in C++ class defines the access control rules. C++ has 3 new keywords introduced, namely public, private, protected. These access specifiers are used to set boundaries for availability of members of class be it data members or member functions. Access specifiers in the program, are followed by a colon. You can use either one,two or all three specifiers in the same class to set different boundaries for different class members. They change the boundary for all the declarations that follow them.

**Static Data Members:**

A data member of a class can be made as static. A static member variable has certain special characteristics.

* It is initialized to zero when the first object of its class is created. No other initialization is permitted.
* Only one copy of that member is created for the entire class and is shared by all the objects of the class , no matter how many object are created.
* It is visible only within the class, but its lifetime is the entire program.

Static variable are used to maintain value common to the entire class. Each static variable must be defined outside the class definition. Static data members are stored separately rather than as a part of an object. Since, they are associated with the class itself rather than with any class object, they are also known as class variables.

**Static Member Functions:**

Like static member variable, we can create static member functions. A member functions that is declared static has the following properties:

1. A static function can have access to only other static members (functions or variables) declared in the same class.
2. A static member functions can be called using the classname instead of its objects as follows:

*class\_name::function\_name;*

**Friend Function:**

C++ allows the common function to be made friendly with the classes, by allowing the function to have access to private data of that class. Such function need not be a member of any of these classes .

Syntax:

*friend data\_type function\_name(class\_name object\_name)*

The function declaration should be preceded by the keyword friend. The function is defined elsewhere in the program like a normal C++ function. The function does not use either the keyword friend or the scope operator :: The functions that are declared with the keyword friend are known as friend functions. A function can be declared as a friend in any number of classes. A friend function has full access rights to the private members of the class. Friend Function can be declared as public or private.

**Practical Related Questions:**

1. Explain memory allocation for objects.
2. Write the difference between structure and class.
3. Write the characteristics of friend functions.
4. Write in brief about arrays of objects.

**Programs :**

1. Develop a class BankAccount with properties like account number, balance, and owner name. Accept and display this data for n number of employees.
2. Write a program to declare the class staff having data members name , department and post. Accept data for 4 staffs and display name of staff who is HOD.
3. Create a class Employee with properties like name, department, salary, and ID. Accept and display this data for 5 employees.

**Conclusion :**

Hence, we learnt about the concept of Class and Objects in CPP.

**Programs**

1. **Develop a class BankAccount with properties like account number, balance, and owner name. Accept and display this data for n number of employees.**

#include <iostream.h>

#include <string.h>

#include <conio.h>

class BankAccount

{

public:

int accountNumber;

double balance;

string ownerName;

void acceptDetails()

{

cout << "Enter Account Number: ";

cin >> accountNumber;

cout << "Enter Owner Name: ";

cin>>ownerName;

cout << "Enter Balance: ";

cin >> balance;

}

void displayDetails()

{

cout << "Account Number: " << accountNumber << endl;

cout << "Owner Name: " << ownerName << endl;

cout << "Balance: " << balance << endl;

}

};

void main()

{

clrscr();

int n;

cout << "Enter the number of employees: ";

cin >> n;

BankAccount\* accounts = new BankAccount[n];

for (int i = 0; i < n; ++i)

{

cout << "\nEnter details for employee " << i + 1 << endl;

accounts[i].acceptDetails();

}

cout << "\nDisplaying account details for all employees:\n";

for (int i = 0; i < n; ++i)

{

cout << "\nEmployee " << i + 1 << " details:" << endl;

accounts[i].displayDetails();

}

getch();

}

1. **Write a program to declare the class staff having data members name , department and post. Accept data for 4 staffs and display name of staff who is HOD.**

#include<iostream.h>

#include<conio.h>

#include<string.h>

class staff

{

public:

char name[30],post[10],department[10];

void getdata( );

void displaydata( );

};

void staff : : getdata( )

{

cout<< ”enter the name of the staff ”<<endl;

cin>>name;

cout<< ”enter the post of the staff ”<<endl;

cin>>post;

cout<< ”enter the department of the staff ”<<endl;

cin>>department;

}

void staff : : displaydata( )

{

if(strcmp(post, ”HOD”) = =0 )

{

cout<< ”The name of the staff is ”<<name<<endl;

cout<< ”The post of the staff is”<<post<<endl;

cout<< ”The department of the staff is ”<<department<<endl;

}

}

void main( )

{

staff s[5];

clrscr( );

for(int i=0 ; i< 5 ; i++)

{

s[i].getdata( );

}

for(int i=0 ; i< 5 ; i++)

{

s[i].displaydata( );

}

getch( );

}

1. **Create a class Employee with properties like name, department, salary, and ID. Accept and display this data for 5 employees.**

#include<iostream.h>

#include<conio.h>

#include<string.h>

class employee

{

public:

int id;

float salary;

char name[30],department[10];

void getdata( );

void displaydata( );

};

void staff : : getdata( )

{

cout<< ”enter the ID of the employee ”<<endl;

cin>>id;

cout<< ”enter the name of the employee ”<<endl;

cin>>name;

cout<< ”enter the salary of the employee ”<<endl;

cin>>salary;

cout<< ”enter the department of the employee ”<<endl;

cin>>department;

}

void staff : : displaydata( )

{

cout<< ”The ID of the employee is ”<<name<<endl;

cout<< ”The name of the employee is ”<<name<<endl;

cout<< ”The salary of the employee is”<<salary<<endl;

cout<< ”The department of the employee is ”<<department<<endl;

}

}

void main( )

{

employee e[5];

clrscr( );

for(int i=0 ; i< 5 ; i++)

{

e[i].getdata( );

}

for(int i=0 ; i< 5 ; i++)

{

e[i].displaydata( );

}

getch( );

}

**Experiment No 9**

**Aim:** To introduce the concept of LaTex.

**Theory:**

# What is LATEX?

LATEX (pronounced “LAY-tek” or “LAH-tek”) is a tool for typesetting professional-looking documents. However, LaTeX’s mode of operation is quite different to many other document-production applications you may have used, such as Microsoft Word or LibreOffice Writer: those “[WYSIWYG](https://en.wikipedia.org/wiki/WYSIWYG)” tools provide users with an interactive page into which they type and edit their text and apply various forms of styling. LaTeX works very differently: instead, your document is a plain text file interspersed with LaTeX commands used to express the desired (typeset) results. To produce a visible, typeset document, your LaTeX file is processed by a piece of software called a TeX engine which uses the commands embedded in your text file to guide and control the typesetting process, converting the LaTeX commands and document text into a professionally typeset PDF file. This means you only need to focus on the content of your document and the computer, via LaTeX commands and the TeX engine, will take care of the visual appearance (formatting).

# Why learn LATEX?

Various arguments can be proposed for, or against, learning to use LATEX instead of other document-authoring applications; but, ultimately, it is a personal choice based on preferences, affinities, and documentation requirements.

Arguments in favour of LATEX include:

* support for typesetting extremely complex mathematics, tables and technical content for the physical sciences;
* facilities for footnotes, cross-referencing and management of bibliographies;
* ease of producing complicated, or tedious, document elements such as indexes, glossaries, table of contents, lists of figures;
* being highly customizable for bespoke document production due to its intrinsic programmability and extensibility through thousands of [free add-on packages](https://www.ctan.org/pkg).

Overall, LATEX provides users with a great deal of control over the production of documents which are typeset to extremely high standards. Of course, there are types of documents or publications where LATEX doesn’t shine, including many “free form” page designs typically found in magazine-type publications.

One important benefit of LATEX is the separation of document content from document style: once you have written the content of your document, its appearance can be changed with ease. Similarly, you can create a LATEX file which defines the layout/style of a particular document type and that file can be used as a template to standardise authorship/production of additional documents of that type; for example, this allows scientific publishers to create article templates, in LATEX, which authors use to write papers for submission to journals. Overleaf has a [gallery containing thousands of templates](https://www.overleaf.com/gallery), covering an enormous range of document types—everything from scientific articles, reports and books to CVs and presentations. Because these templates define the layout and style of the document, authors need only to open them in Overleaf—creating a new project—and commence writing to add their content.

# Writing your first piece of LATEX

The first step is to create a new LATEX project. You can do this on your own computer by creating a new .tex file.

Let’s start with the simplest example:

\documentclass{article}

\begin{document}

First document. This is a simple example, with no

extra parameters or packages included.

\end{document}

The first line of code, \documentclass{article}, declares the document type known as its class, which controls the overall appearance of the document. Different types of documents require different classes; i.e., a CV/resume will require a different class than a scientific paper which might use the standard LATEX article class. Other types of documents you may be working on may require different classes such as book or report.

Having set the document class, our content, known as the body of the document, is written between the \begin{document} and \end{document} tags. After opening the example above, you can make changes to the text and, when finished, view the resulting typeset PDF by recompiling the document*.*

**The preamble of a document**

The previous example showed how document content was entered after the \begin{document} command; however, everything in your .tex file appearing before that point is called the preamble, which acts as the document’s “setup” section. Within the preamble you define the document class (type) together with specifics such as languages to be used when writing the document; loading packages you would like to use (more on this [later](https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes" \l "Finding_and_using_LaTeX_packages)), and it is where you’d apply other types of configuration.

A minimal document preamble might look like this:

\documentclass[12pt, letterpaper]{article}

\usepackage{graphicx}

where \documentclass[12pt, letterpaper]{article} defines the overall class (type) of document. Additional parameters, which must be separated by commas, are included in square brackets ([...]) and used to configure this instance of the article class; i.e., settings we wish to use for this particular article-class-based document.

In this example, the two parameters do the following:

* 12pt sets the font size
* letterpaper sets the paper size

Of course other font sizes, 9pt, 11pt, 12pt, can be used, but if none is specified, the default size is 10pt. As for the paper size, other possible values are a4paper and legalpaper. For further information see the article about [page size and margins](https://www.overleaf.com/learn/latex/Page_size_and_margins" \o "Page size and margins).

The preamble line

\usepackage{graphicx}

is an example of loading an external package (here, [graphicx](https://ctan.org/pkg/graphicx?lang=en)) to extend LATEX’s capabilities, enabling it to import external graphics files.

**Including title, author and date information**

Adding a title, author and date to our document requires three more lines in the preamble (not the main body of the document). Those lines are:

* \title{My first LaTeX document}: the document title
* \author{Hubert Farnsworth}: here you write the name of the author(s) and, optionally, the \thanks command within the curly braces:
  + \thanks{Funded by the Overleaf team.}: can be added after the name of the author, inside the braces of the author command. It will add a superscript and a footnote with the text inside the braces. Useful if you need to thank an institution in your article.
* \date{August 2022}: you can enter the date manually or use the command \today to typeset the current date every time the document is compiled

With these lines added, your preamble should look something like this:

\documentclass[12pt, letterpaper]{article}

\title{My first LaTeX document}

\author{Hubert Farnsworth\thanks{Funded by the Overleaf team.}}

\date{August 2022}

To typeset the title, author and date use the \maketitle command within the body of the document:

\begin{document}

\maketitle

We have now added a title, author and date to our first \LaTeX{} document!

\end{document}

**Adding comments**

LaTeX is a form of “program code”, but one which specializes in document typesetting; consequently, as with code written in any other programming language, it can be very useful to include comments within your document. A LATEX comment is a section of text that will not be typeset or affect the document in any way—often used to add “to do” notes; include explanatory notes; provide in-line explanations of tricky macros or comment-out lines/sections of LaTeX code when debugging.

To make a comment in LATEX, simply write a % symbol at the beginning of the line, as shown in the following code which uses the example above:

\documentclass[12pt, letterpaper]{article}

\title{My first LaTeX document}

\author{Hubert Farnsworth\thanks{Funded by the Overleaf team.}}

\date{August 2022}

\begin{document}

\maketitle

We have now added a title, author and date to our first \LaTeX{} document!

% This line here is a comment. It will not be typeset in the document.

\end{document}

**Practical Related Questions:**

1. How does LaTeX works?
2. Write the advantages of LaTex.
3. What is ”The Preamble of a Document”. Write its sample code with font size as 16 and paper size as A4.

**Experiment No 10**

**Aim:** To study how to edit text and insert list and images in a document.

**Theory:**

# Bold, italics and underlining

We will now look at some text formatting commands:

* **Bold**: bold text in LaTeX is typeset using the \textbf{...} command.
* *Italics*: italicised text is produced using the \textit{...} command.
* Underline: to underline text use the \underline{...} command.

The next example demonstrates these commands:

Some of the \textbf{greatest}

discoveries in \underline{science}

were made by \textbf{\textit{accident}}.

Another very useful command is \emph{argument}, whose effect on its argument depends on the context. Inside normal text, the emphasized text is italicized, but this behaviour is reversed if used inside an italicized text—see the next example:

Some of the greatest \emph{discoveries} in science were made by accident.

\textit{Some of the greatest \emph{discoveries} in science were made by accident.}

\textbf{Some of the greatest \emph{discoveries} in science were made by accident.}

# Adding images

The following example demonstrates how to include a picture:

\documentclass{article}

\usepackage{graphicx} %LaTeX package to import graphics

\graphicspath{{images/}} %configuring the graphicx package

\begin{document}

The universe is immense and it seems to be homogeneous, on a large scale, everywhere we look.

% The \includegraphcs command is

% provided (implemented) by the

% graphicx package

\includegraphics{universe}

There's a picture of a galaxy above.

\end{document}

Importing graphics into a LATEX document needs [an add-on package](https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes" \l "Finding_and_using_LaTeX_packages) which provides the commands and features required to include external graphics files. The above example loads the [graphicx package](https://ctan.org/pkg/graphicx?lang=en) which, among many other commands, provides \includegraphics{...} to import graphics and \graphicspath{...} to advise LATEX where the graphics are located.

To use the graphicx package, include the following line in your Overleaf document preamble:

\usepackage{graphicx}

The \includegraphics{universe} command does the actual work of inserting the image in the document. Here, universe is the name of the image file but without its extension.

## Captions, labels and references

Images can be captioned, labelled and referenced by means of the figure environment, as shown below:

\documentclass{article}

\usepackage{graphicx}

\graphicspath{{images/}}

\begin{document}

\begin{figure}[h]

\centering

\includegraphics[width=0.75\textwidth]{mesh}

\caption{A nice plot.}

\label{fig:mesh1}

\end{figure}

As you can see in figure \ref{fig:mesh1}, the function grows near the origin. This example is on page \pageref{fig:mesh1}.

\end{document}

There are several noteworthy commands in the example:

* **\includegraphics[width=0.75\textwidth]{mesh}**: This form of \includegraphics instructs LATEX to set the figure’s width to 75% of the text width—whose value is stored in the \textwidth command.
* **\caption{A nice plot.}**: As its name suggests, this command sets the figure caption which can be placed above or below the figure. If you create a list of figures this caption will be used in that list.
* **\label{fig:mesh1}**: To reference this image within your document you give it a label using the \label command. The label is used to generate a number for the image and, combined with the next command, will allow you to reference it.
* **\ref{fig:mesh1}**: This code will be substituted by the number corresponding to the referenced figure.

Images incorporated in a LATEX document should be placed inside a figure environment, or similar, so that LATEX can automatically position the image at a suitable location in your document.

# Creating lists in LATEX

You can create different types of list using environments, which are used to encapsulate the LATEX code required to implement a specific typesetting feature. An environment starts with \begin{environment-name} and ends with \end{environment-name} where environment-name might be figure, tabular or one of the list types: itemize for unordered lists or enumerate for ordered lists.

## Unordered lists

Unordered lists are produced by the itemize environment. Each list entry must be preceded by the \item command, as shown below:

\documentclass{article}

\begin{document}

\begin{itemize}

\item The individual entries are indicated with a black dot, a so-called bullet.

\item The text in the entries may be of any length.

\end{itemize}

\end{document}

## Ordered lists

Ordered lists use the same syntax as unordered lists but are created using the enumerate environment:

\documentclass{article}

\begin{document}

\begin{enumerate}

\item This is the first entry in our list.

\item The list numbers increase with each entry we add.

\end{enumerate}

\end{document}

**Practical Related Questions:**

1. How to add list into the document? Write an example to add an unordered list with all the subjects of your this semester and ordered list with the names of your friends.
2. Write the code to add multiple figure in the image.

**Experiment No 11**

**Aim:** To study how to edit Math Code/Mathematical Equations and insert into document.

**Theory:**

**Adding math to LATEX**

One of the main advantages of LATEX is the ease with which mathematical expressions can be written. LATEX provides two writing modes for typesetting mathematics:

* inline math mode used for writing formulas that are part of a paragraph
* display math mode used to write expressions that are not part of a text or paragraph and are typeset on separate lines

## Inline math mode

Let’s see an example of inline math mode:

\documentclass[12pt, letterpaper]{article}

\begin{document}

In physics, the mass-energy equivalence is stated

by the equation $E=mc^2$, discovered in 1905 by Albert Einstein.

\end{document}

To typeset inline-mode math you can use one of these delimiter pairs: \( ... \), $ ... $ or \begin{math} ... \end{math}, as demonstrated in the following example:

\documentclass[12pt, letterpaper]{article}

\begin{document}

\begin{math}

E=mc^2

\end{math} is typeset in a paragraph using inline math mode---as is $E=mc^2$, and so too is \(E=mc^2\).

\end{document}

## Display math mode

Equations typeset in display mode can be numbered or unnumbered, as in the following example:

\documentclass[12pt, letterpaper]{article}

\begin{document}

The mass-energy equivalence is described by the famous equation

\[ E=mc^2 \] discovered in 1905 by Albert Einstein.

In natural units ($c = 1$), the formula expresses the identity

\begin{equation}

E=m

\end{equation}

\end{document}

To typeset display-mode math you can use one of these delimiter pairs: \[ ... \], \begin{displaymath} ... \end{displaymath} or \begin{equation} ... \end{equation}. Historically, typesetting display-mode math required use of $$ characters delimiters, as in $$ ... display math here ...$$, but this method [is no longer recommended](https://texfaq.org/FAQ-dolldoll): use LaTeX’s delimiters \[ ... \] instead.

## More complete examples

The following examples demonstrate a range of mathematical content typeset using LaTeX.

\documentclass{article}

\begin{document}

Subscripts in math mode are written as $a\_b$ and superscripts are written as $a^b$. These can be combined and nested to write expressions such as

\[ T^{i\_1 i\_2 \dots i\_p}\_{j\_1 j\_2 \dots j\_q} = T(x^{i\_1},\dots,x^{i\_p},e\_{j\_1},\dots,e\_{j\_q}) \]

We write integrals using $\int$ and fractions using $\frac{a}{b}$. Limits are placed on integrals using superscripts and subscripts:

\[ \int\_0^1 \frac{dx}{e^x} = \frac{e-1}{e} \]

Lower case Greek letters are written as $\omega$ $\delta$ etc. while upper case Greek letters are written as $\Omega$ $\Delta$.

Mathematical operators are prefixed with a backslash as $\sin(\beta)$, $\cos(\alpha)$, $\log(x)$ etc.

\end{document}

The next example uses the equation\* environment which is provided by the amsmath package, so we need to add the following line to our document preamble:

\usepackage{amsmath}% For the equation\* environment

For further information on using amsmath see [our help article](https://www.overleaf.com/learn/latex/Aligning_equations" \o "Aligning equations).

\documentclass{article}

\usepackage{amsmath}% For the equation\* environment

\begin{document}

\section{First example}

The well-known Pythagorean theorem \(x^2 + y^2 = z^2\) was proved to be invalid for other exponents, meaning the next equation has no integer solutions for \(n>2\):

\[ x^n + y^n = z^n \]

\section{Second example}

This is a simple math expression \(\sqrt{x^2+1}\) inside text.

And this is also the same:

\begin{math}

\sqrt{x^2+1}

\end{math}

but by using another command.

This is a simple math expression without numbering

\[\sqrt{x^2+1}\]

separated from text.

This is also the same:

\begin{displaymath}

\sqrt{x^2+1}

\end{displaymath}

\ldots and this:

\begin{equation\*}

\sqrt{x^2+1}

\end{equation\*}

\end{document}

**Practical Related Questions:**

1. Write the code to print the examples of subscript and the superscript.
2. Write the code to print 10 different mathematical equations.

**Experiment No 12**

**Aim:** To create a document structure in latex by adding tables and packages

**Theory:**

# Basic document structure

# We explore abstracts and how to partition a LATEX document into different chapters, sections and paragraphs.

## Abstracts

Scientific articles usually provide an abstract which is a brief overview/summary of their core topics, or arguments. The next example demonstrates typesetting an abstract using LATEX’s abstract environment:

\documentclass{article}

\begin{document}

\begin{abstract}

This is a simple paragraph at the beginning of the

document. A brief introduction about the main subject.

\end{abstract}

\end{document}

## Paragraphs and new lines

With the abstract in place, we can begin writing our first paragraph. The next example demonstrates:

* how a new paragraph is created by pressing the "enter" key twice, ending the current line and inserting a subsequent blank line;
* how to start a new line without starting a new paragraph by inserting a manual line break using the \\ command, which is a double backslash; alternatively, use the \newline command.

The third paragraph in this example demonstrates use of the commands \\ and \newline:

\documentclass{article}

\begin{document}

\begin{abstract}

This is a simple paragraph at the beginning of the

document. A brief introduction about the main subject.

\end{abstract}

After our abstract we can begin the first paragraph, then press ``enter'' twice to start the second one.

This line will start a second paragraph.

I will start the third paragraph and then add \\ a manual line break which causes this text to start on a new line but remains part of the same paragraph. Alternatively, I can use the \verb|\newline|\newline command to start a new line, which is also part of the same paragraph.

\end{document}

## Chapters and sections

Longer documents, irrespective of authoring software, are usually partitioned into parts, chapters, sections, subsections and so forth. LaTeX also provides document-structuring commands but the available commands, and their implementations (what they do), can depend on the document class being used. By way of example, documents created using the book class can be split into parts, chapters, sections, subsections and so forth but the letter class does not provide (support) any commands to do that.

This next example demonstrates commands used to structure a document based on the book class:

\documentclass{book}

\begin{document}

\chapter{First Chapter}

\section{Introduction}

This is the first section.

\section{Second Section}

\subsection{First Subsection}

\section\*{Unnumbered Section}

Lorem ipsum dolor sit amet, consectetuer adipiscing elit.

Etiam lobortis facilisissem...

\end{document}

The names of sectioning commands are mostly self-explanatory; for example, \chapter{First Chapter} creates a new chapter titled First Chapter, \section{Introduction} produces a section titled Introduction, and so forth. Sections can be further divided into \subsection{...} and even \subsubsection{...}. The numbering of sections, subsections etc. is automatic but can be disabled by using the so-called starred version of the appropriate command which has an asterisk (\*) at the end, such as \section\*{...} and \subsection\*{...}.

Collectively, LaTeX document classes provide the following sectioning commands, with specific classes each supporting a relevant subset:

* \part{part}
* \chapter{chapter}
* \section{section}
* \subsection{subsection}
* \subsubsection{subsubsection}
* \paragraph{paragraph}
* \subparagraph{subparagraph}

In particular, the \part and \chapter commands are only available in the report and book document classes.

Visit the Overleaf article [article about sections and chapters](https://www.overleaf.com/learn/latex/Sections_and_chapters" \o "Sections and chapters) for further information about document-structure commands.

# Creating tables

The following examples show how to create tables in LaTeX, including the addition of lines (rules) and captions.

## Creating a basic table in LATEX

We start with an example showing how to typeset a basic table:

\begin{center}

\begin{tabular}{c c c}

cell1 & cell2 & cell3 \\

cell4 & cell5 & cell6 \\

cell7 & cell8 & cell9

\end{tabular}

\end{center}

The tabular environment is the default LATEX method to create tables. You must specify a parameter to this environment, in this case {c c c} which advises LATEX that there will be three columns and the text inside each one must be centred. You can also use r to right-align the text and l to left-align it. The alignment symbol & is used to demarcate individual table cells within a table row. To end a table row use the *new line* command \\. Our table is contained within a center environment to make it centred within the text width of the page.

## Adding borders

The tabular environment supports horizontal and vertical lines (rules) as part of the table:

* to add horizontal rules, above and below rows, use the \hline command
* to add vertical rules, between columns, use the vertical line parameter |

In this example the argument is {|c|c|c|} which declares three (centred) columns each separated by a vertical line (rule); in addition, we use \hline to place a horizontal rule above the first row and below the final row:

\begin{center}

\begin{tabular}{|c|c|c|}

\hline

cell1 & cell2 & cell3 \\

cell4 & cell5 & cell6 \\

cell7 & cell8 & cell9 \\

\hline

\end{tabular}

\end{center}

Here is a further example:

\begin{center}

\begin{tabular}{||c c c c||}

\hline

Col1 & Col2 & Col2 & Col3 \\ [0.5ex]

\hline\hline

1 & 6 & 87837 & 787 \\

\hline

2 & 7 & 78 & 5415 \\

\hline

3 & 545 & 778 & 7507 \\

\hline

4 & 545 & 18744 & 7560 \\

\hline

5 & 88 & 788 & 6344 \\ [1ex]

\hline

\end{tabular}

\end{center}

## Captions, labels and references

You can caption and reference tables in much the same way as images. The only difference is that instead of the figure environment, you use the table environment.

Table \ref{table:data} shows how to add a table caption and reference a table.

\begin{table}[h!]

\centering

\begin{tabular}{||c c c c||}

\hline

Col1 & Col2 & Col2 & Col3 \\ [0.5ex]

\hline\hline

1 & 6 & 87837 & 787 \\

2 & 7 & 78 & 5415 \\

3 & 545 & 778 & 7507 \\

4 & 545 & 18744 & 7560 \\

5 & 88 & 788 & 6344 \\ [1ex]

\hline

\end{tabular}

\caption{Table to test captions and labels.}

\label{table:data}

\end{table}

# Adding a Table of Contents

Creating a table of contents is straightforward because the command \tableofcontents does almost all the work for you:

\documentclass{article}

\title{Sections and Chapters}

\author{Gubert Farnsworth}

\date{August 2022}

\begin{document}

\maketitle

\tableofcontents

\section{Introduction}

This is the first section.

\section\*{Unnumbered Section}

\addcontentsline{toc}{section}{Unnumbered Section}

\section{Second Section}

\end{document}

Sections, subsections and chapters are automatically included in the table of contents. To manually add entries, such as an unnumbered section, use the command \addcontentsline as shown in the example.

LATEX not only delivers significant typesetting capabilities but also provides a framework for extensibility through the use of add-on packages. Rather than attempting to provide commands and features that “try to do everything”, LATEX is designed to be extensible, allowing users to load external bodies of code (packages) that provide more specialist typesetting capabilities or extend LATEX’s built-in features—such as typesetting tables. As observed in the section [Adding images](https://www.overleaf.com/learn/latex/Learn_LaTeX_in_30_minutes" \l "Adding_images), the graphicx package extends LATEX by providing commands to import graphics files and was loaded (in the preamble) by writing

\usepackage{graphicx}

## Loading packages

As noted above, packages are loaded in the document preamble via the \usepackage command but because (many) LATEX packages provide a set of options, which can be used to configure their behaviour, the \usepackage command often looks like this:

\usepackage[options]{somepackage}

The square brackets “[...]” inform LATEX which set of options should be applied when it loads somepackage. Within the set of options requested by the user, individual options, or settings, are typically separated by a comma; for example, the [geometry package](https://ctan.org/pkg/geometry) provides many options to configure page layout in LATEX, so a typical use of geometry might look like this:

\usepackage[total={6.5in,8.75in},

top=1.2in, left=0.9in, includefoot]{geometry}

The geometry package is one example of a package written and contributed by members of the global LATEX community and made available, for free, to anyone who wants to use it.

If a LATEX package does not provide any options, or the user wants to use the default values of a package’s options, it would be loaded like this:

\usepackage{somepackage}

When you write \usepackage[...]{somepackage} LATEX looks for a corresponding file called somepackage.sty, which it needs to load and process—to make the package commands available and execute any other code provided by that package. If LATEX cannot find somepackage.sty it will terminate with an error, as demonstrated in the following Overleaf example:

\documentclass[12pt, letterpaper]{article}

\usepackage{somepackage}% a NON-EXISTENT package

\begin{document}

This will fail!

\end{document}

**Practical Related Questions:**

Attach the print of given assignment.

**Experiment No 13**

**Aim:** To learn Basic Git Commands.

**Theory:**

## What is Git?

Git is a popular version control system. It was created by Linus Torvalds in 2005, and has been maintained by Junio Hamano since then.

**It is used for:**

* Tracking code changes
* Tracking who made changes
* Coding collaboration

### What does Git do?

* Manage projects with **Repositories**
* **Clone** a project to work on a local copy
* Control and track changes with **Staging** and **Committing**
* **Branch** and **Merge** to allow for work on different parts and versions of a project
* **Pull** the latest version of the project to a local copy
* **Push** local updates to the main project

### Working with Git

* Initialize Git on a folder, making it a **Repository**
* Git now creates a hidden folder to keep track of changes in that folder
* When a file is changed, added or deleted, it is considered **modified**
* You select the modified files you want to **Stage**
* The **Staged** files are **Committed**, which prompts Git to store a **permanent** snapshot of the files
* Git allows you to see the full history of every commit.
* You can revert back to any previous commit.
* Git does not store a separate copy of every file in every commit, but keeps track of changes made in each commit!

## Why Git?

* Over 70% of developers use Git!
* Developers can work together from anywhere in the world.
* Developers can see the full history of the project.
* Developers can revert to earlier versions of a project.

### What is GitHub?

* Git is not the same as GitHub.
* GitHub makes tools that use Git.
* GitHub is the largest host of source code in the world, and has been owned by Microsoft since 2018.

## Git Commands: Working With Local Repositories

### git init

* The command git init is used to create an empty Git repository.
* After the git init command is used, a .git folder is created in the directory with some subdirectories. Once the repository is initialized, the process of creating other files begins.

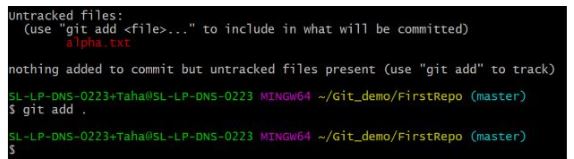
|  |
| --- |
| git init |



### git add

* Add command is used after checking the status of the files, to add those files to the staging area.
* Before running the commit command, "git add" is used to add any new or modified files.

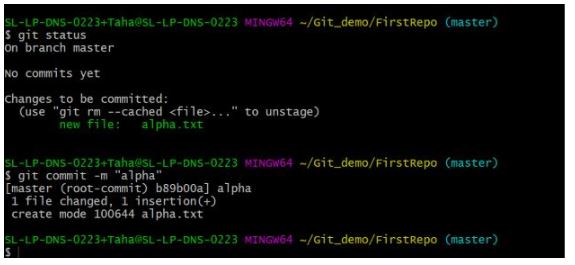
|  |
| --- |
| git add . |



### git commit

* The commit command makes sure that the changes are saved to the local repository.
* The command "git commit –m <message>" allows you to describe everyone and help them understand what has happened.

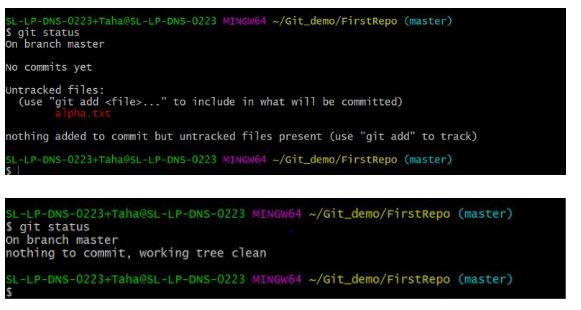
|  |
| --- |
| git commit -m “commit message” |



### git status

* The git status command tells the current state of the repository.
* The command provides the current working branch. If the files are in the staging area, but not committed, it will be shown by the git status. Also, if there are no changes, it will show the message no changes to commit, working directory clean.

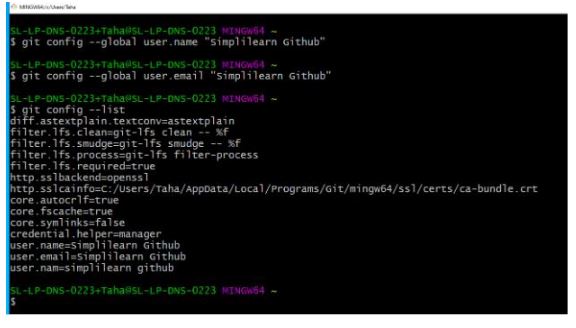
|  |
| --- |
| git status |



### git config

* The git config command is used initially to configure the user.name and user.email. This specifies what email id and username will be used from a local repository.
* When git config is used with --global flag, it writes the settings to all repositories on the computer.

|  |
| --- |
| git config --global user.name “any user name”  git config --global user.email <email id> |



### git branch

* The git branch command is used to determine what branch the local repository is on.
* The command enables adding and deleting a branch.

|  |
| --- |
| # Create a new branch   git branch <branch\_name> |
| # List all remote or local branches   git branch -a |
| # Delete a branch   git branch -d <branch\_name> |

### git checkout

* The git checkout command is used to switch branches, whenever the work is to be started on a different branch.
* The command works on three separate entities: files, commits, and branches.

|  |
| --- |
| # Checkout an existing branch   git checkout <branch\_name> |
| # Checkout and create a new branch with that name   git checkout -b <new\_branch> |

### git merge

* The [git merge](https://www.simplilearn.com/tutorials/git-tutorial/merge-conflicts-in-git" \t "_blank" \o "git merge) command is used to integrate the branches together. The command combines the changes from one branch to another branch.
* It is used to merge the changes in the staging branch to the stable branch.

|  |
| --- |
| git merge <branch\_name> |

**Practical Related Questions:**

1. What is the purpose of the git init command?

2. How do I clone an existing Git repository?

3. What files are added to the staging area using git add?

4. What does git commit do?

5. How do I check the status of my files?

6. What is the purpose of the git log command?

7. How do I compare the current state of a file to the previous commit?

8. What does git checkout do?

9. What does git pull do?

**Experiment No 14**

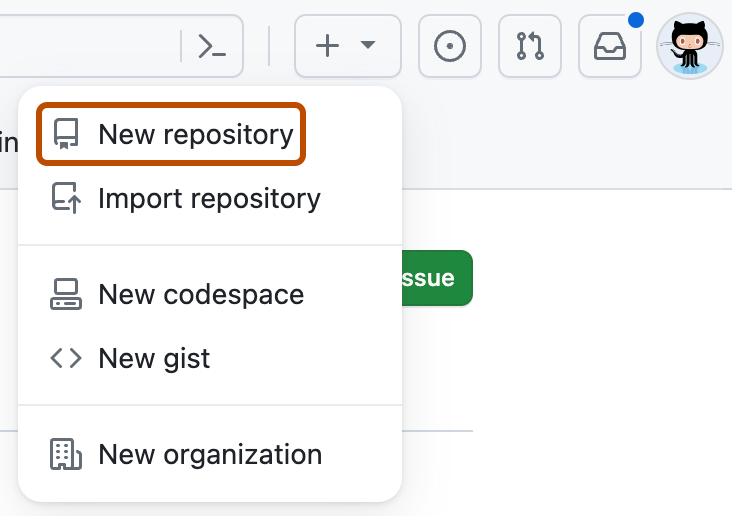
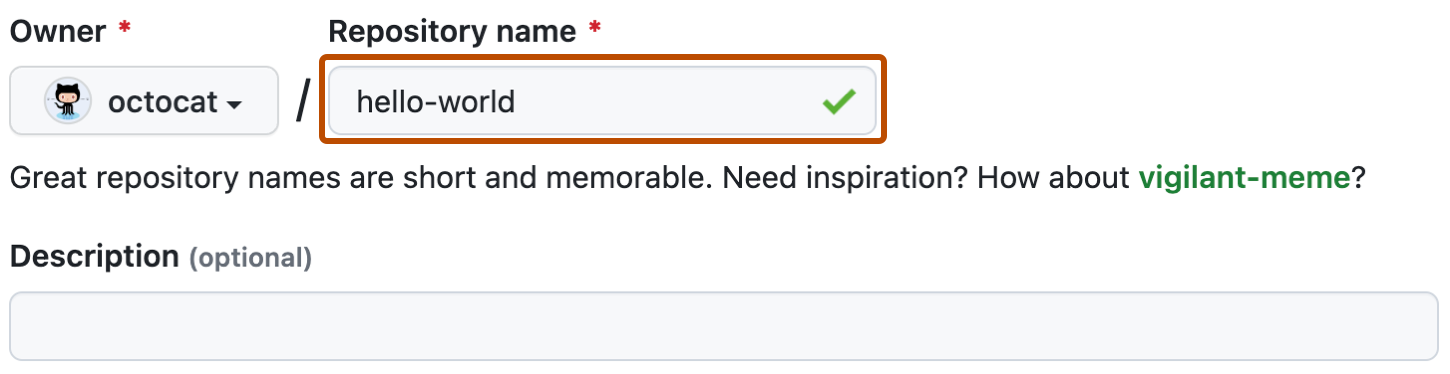
**Aim:** Learning to Create Git repository , creating and editing files, configuring Git.

**Theory:**

# Create a repo

## [Create a repository](https://docs.github.com/en/get-started/quickstart/create-a-repo" \l "create-a-repository)

You can store a variety of projects in GitHub repositories, including open source projects. With open source projects, you can share code to make better, more reliable software. You can use repositories to collaborate with others and track your work. For more information, see "[About repositories](https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories)." To learn more about open source projects, visit [OpenSource.org](https://opensource.org/about).

1. In the upper-right corner of any page, select , then click **New repository**.
   1. 
2. Type a short, memorable name for your repository. For example, "hello-world".
   1. 
3. Optionally, add a description of your repository. For example, "My first repository on GitHub."
4. Choose a repository visibility.
5. Select **Initialize this repository with a README**.
6. Click **Create repository**.

## Creating New Files

# Creating new files

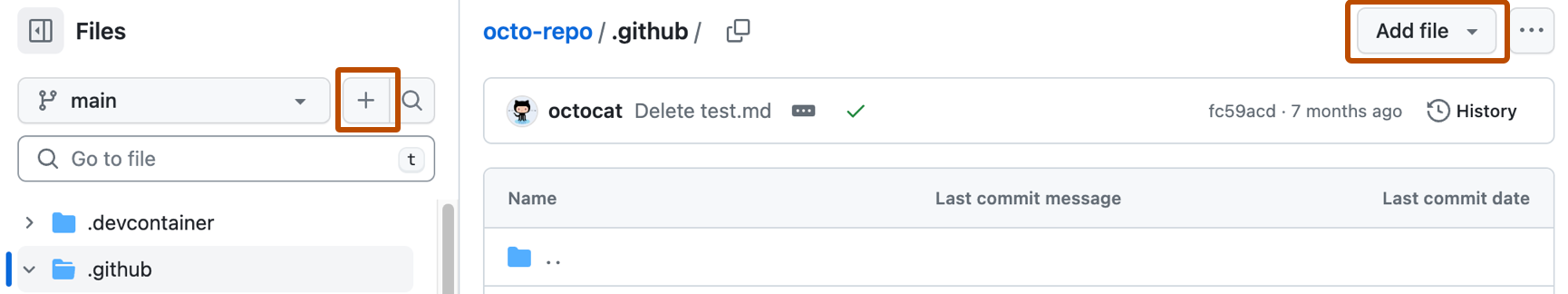
To create new files directly on GitHub in any repository you have write access to.

When creating a file on GitHub, consider the following:

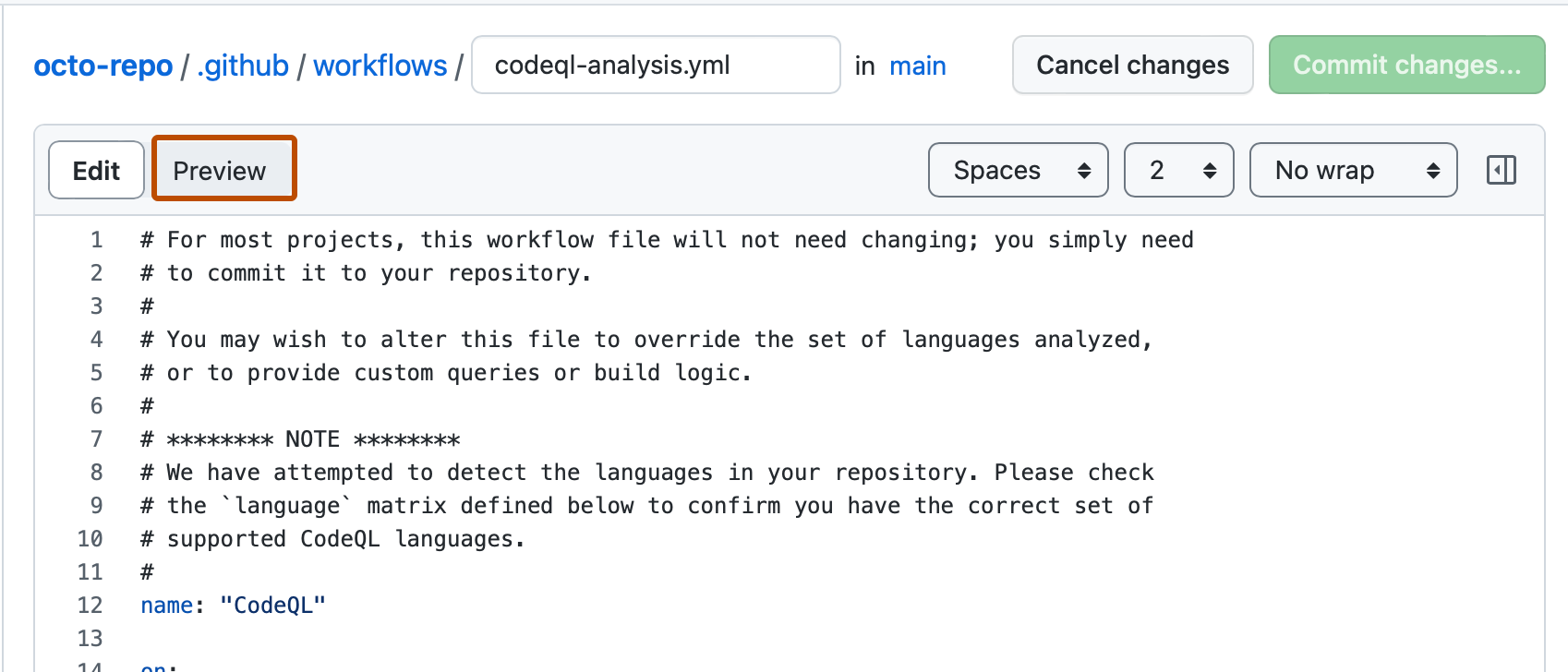
* If you try to create a new file in a repository that you don’t have access to, we will fork the project to your personal account and help you send [a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests) to the original repository after you commit your change.
* File names created via the web interface can only contain alphanumeric characters and hyphens (-). To use other characters, [create and commit the files locally, then push them to the repository on GitHub](https://docs.github.com/en/repositories/working-with-files/managing-files/adding-a-file-to-a-repository).

1. On GitHub.com, navigate to the main page of the repository.
2. In your repository, browse to the folder where you want to create a file.
3. Above the list of files, select the **Add file**  dropdown menu, then click  **Create new file**.

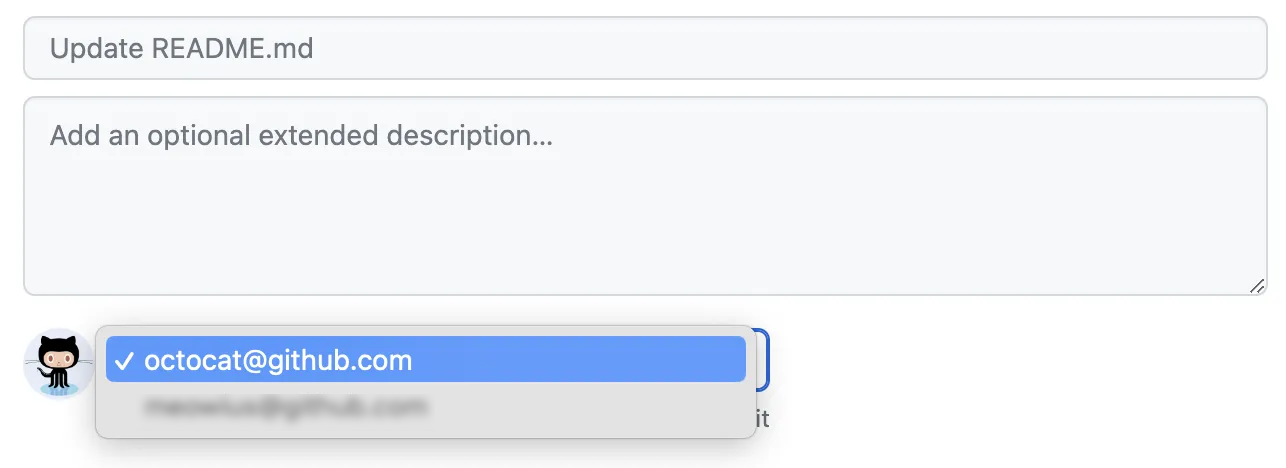
Alternatively, you can click  in the file tree view on the left.



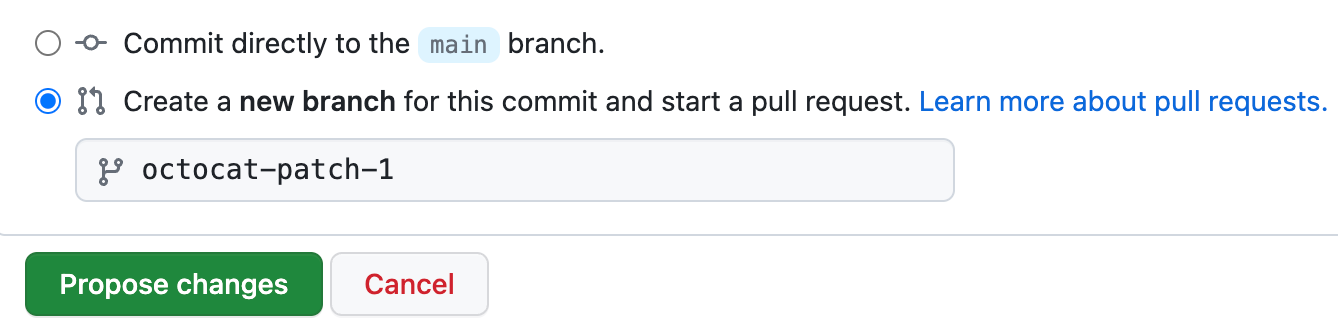
1. In the file name field, type the name and extension for the file. To create subdirectories, type the / directory separator.
2. In the file contents text box, type content for the file.
3. To review the new content, above the file contents, click **Preview**.



1. Click **Commit changes...**
2. In the "Commit message" field, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "[Creating a commit with multiple authors](https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors)."
3. If you have more than one email address associated with your account on GitHub.com, click the email address drop-down menu and select the email address to use as the Git author email address. Only verified email addresses appear in this drop-down menu. If you enabled email address privacy, then a no-reply will be the default commit author email address. For more information about the exact form the no-reply email address can take, see "[Setting your commit email address](https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address)."



1. Below the commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit and then create a pull request. For more information, see "[Creating a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request)."



1. Click **Commit changes** or **Propose changes**.

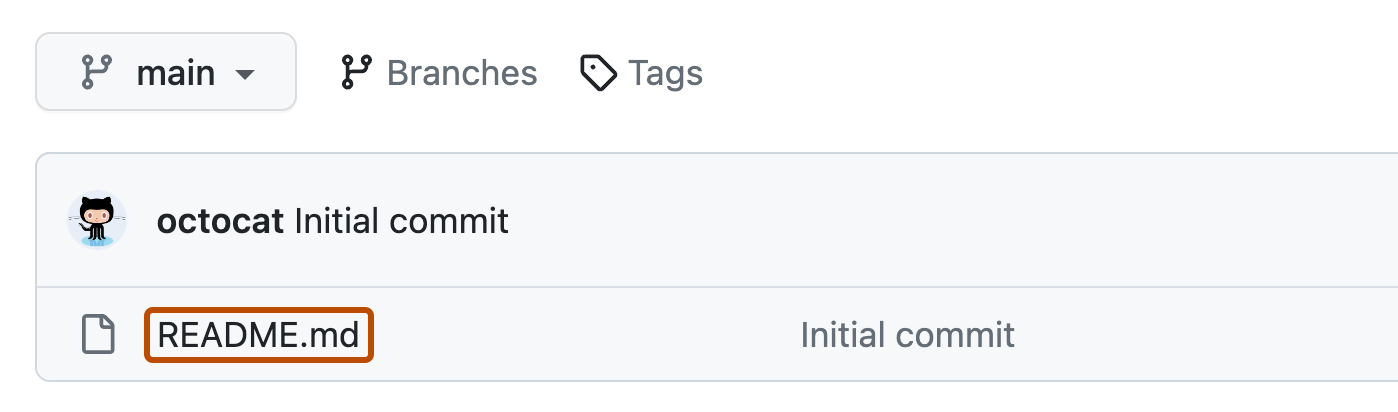
## [Editing](https://docs.github.com/en/get-started/quickstart/create-a-repo" \l "commit-your-first-change) Files

A [commit](https://docs.github.com/en/get-started/quickstart/github-glossary" \l "commit) is like a snapshot of all the files in your project at a particular point in time.

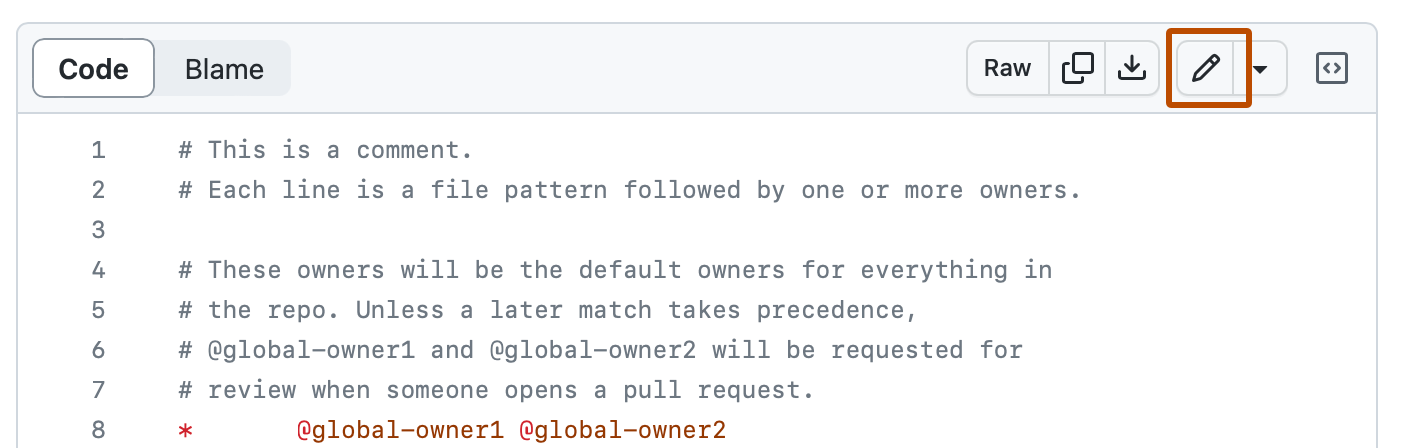
When you created your new repository, you initialized it with a README file. README files are a great place to describe your project in more detail, or add some documentation such as how to install or use your project. The contents of your README file are automatically shown on the front page of your repository.

Let's commit a change to the README file.

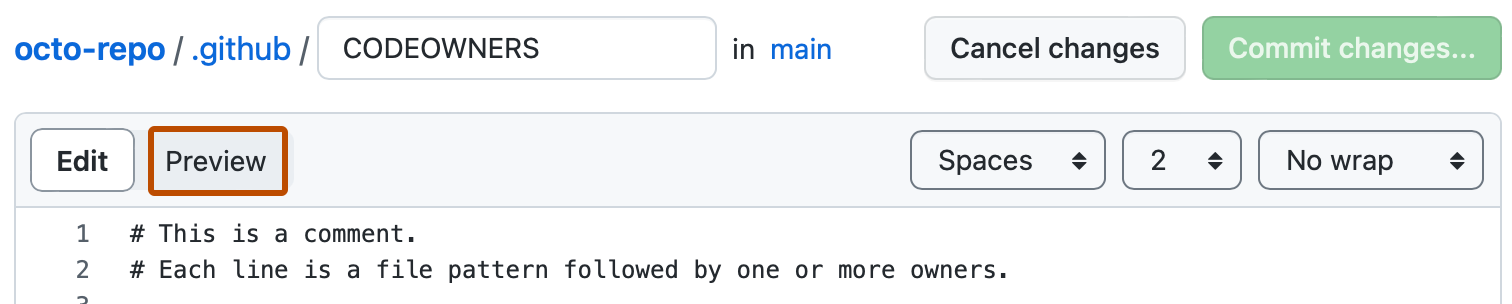
1. In your repository's list of files, select **README.md**.



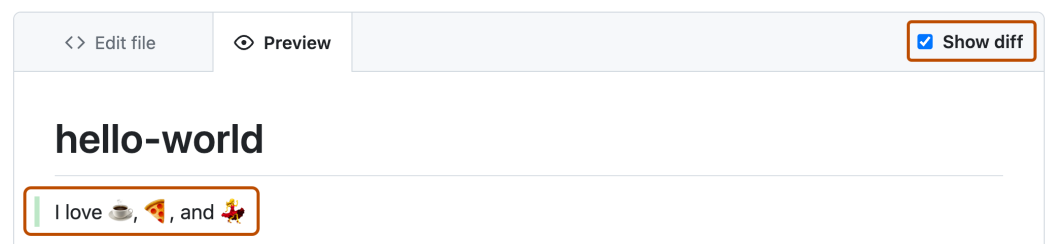
1. In the upper right corner of the file view, click  to open the file editor.



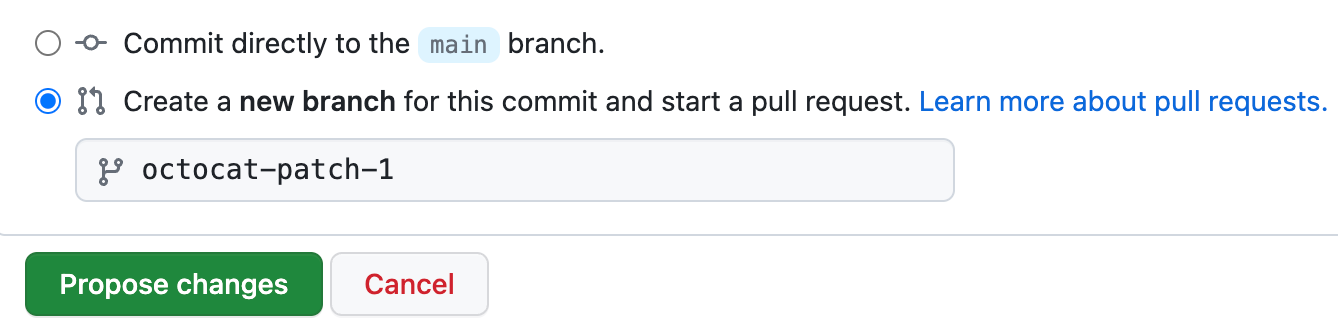
1. In the text box, type some information about yourself.
2. Above the new content, click **Preview**.



1. Review the changes you made to the file. If you select **Show diff**, you will see the new content in green.



1. Click **Commit changes...**
2. In the "Commit message" field, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "[Creating a commit with multiple authors](https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors)."
3. Below the commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit and then create a pull request. For more information, see "[Creating a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request)."



1. Click **Commit changes** or **Propose changes**.

**Practical Related Questions:**

1. What is the purpose of the .gitignore file, and how do you create one?
2. How do you clone an existing Git repository from a remote URL?
3. What are the different ways to add files to a Git repository?
4. What is the purpose of the git status command, and how do you use it?
5. How do you set up SSH keys for secure remote repository access?
6. How do you manage global Git configuration settings?

**Experiment No 15**

**Aim:** Learning Editing Git files and downloading a repository

**Theory:**

# Editing files

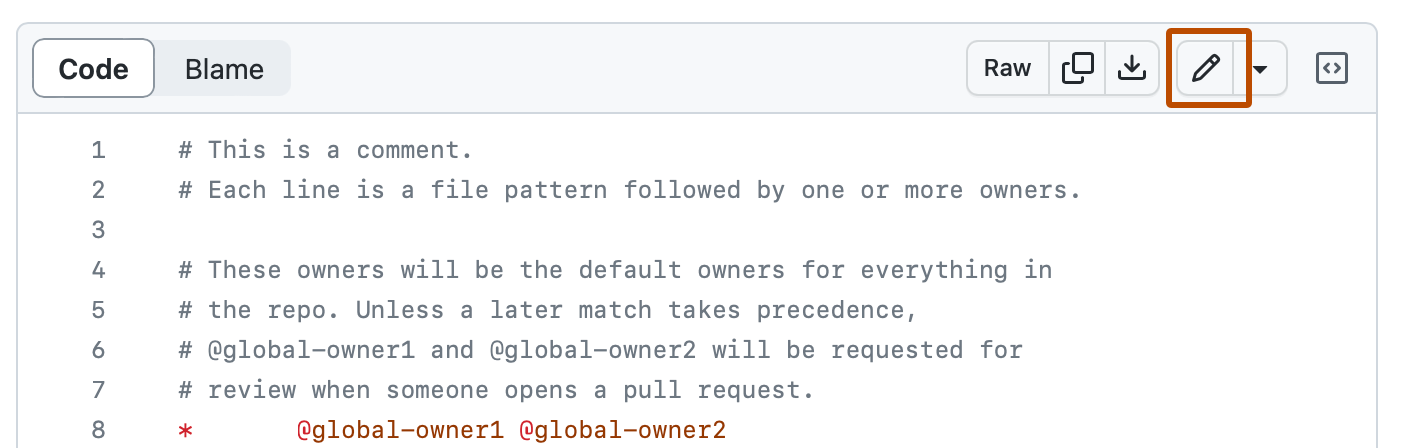
Edit files directly on GitHub in any of your repositories using the file editor.

Editing files in your repository

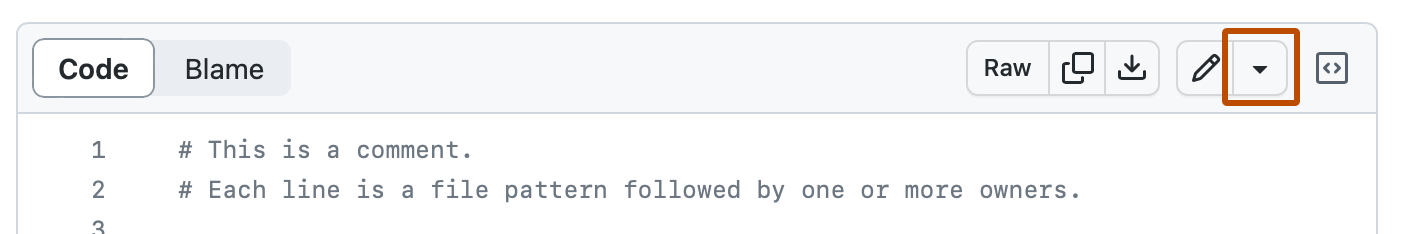
Tip: If a repository has any protected branches, you can't edit or upload files in the protected branch using GitHub. For more information, see "About protected branches."

You can use GitHub Desktop to move your changes to a new branch and commit them.

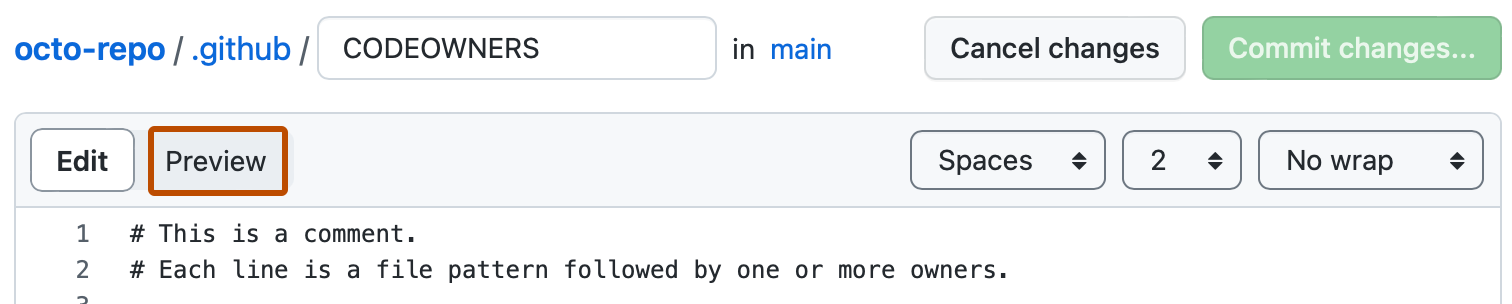
1. n your repository, browse to the file you want to edit.
2. In the upper right corner of the file view, click to open the file editor.



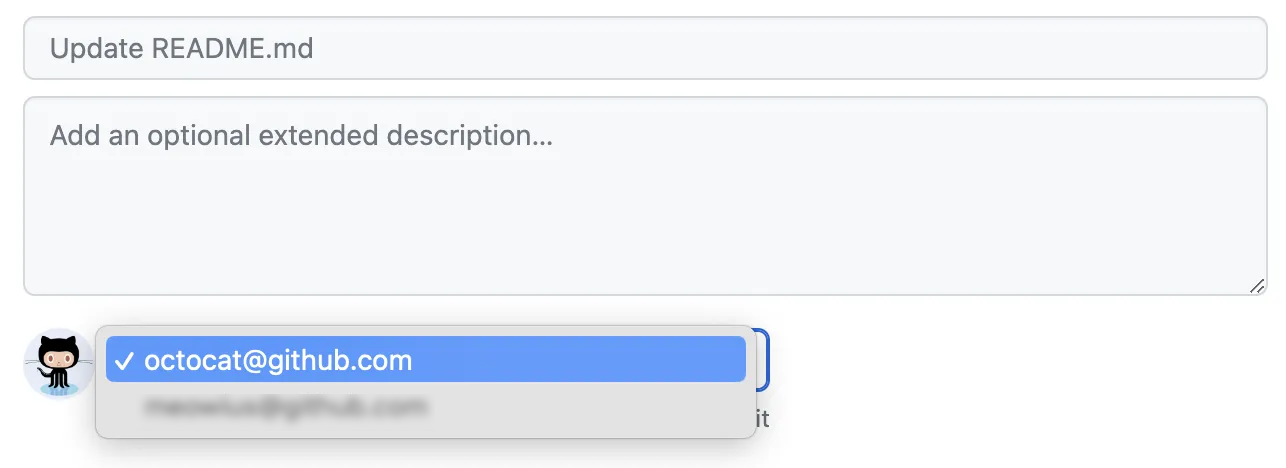
**Note:** Instead of editing and committing the file using the default file editor, you can optionally choose to use the [github.dev code editor](https://docs.github.com/en/codespaces/the-githubdev-web-based-editor) by selecting the  dropdown menu and clicking **github.dev**. You can also clone the repository and edit the file locally via GitHub Desktop by clicking **GitHub Desktop**.



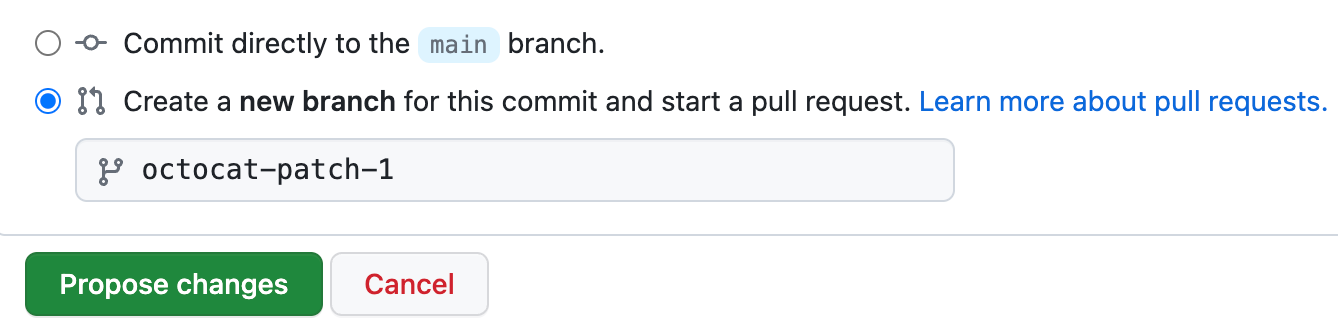
1. In the text box, make any changes you need to the file.
2. Above the new content, click **Preview**.



1. Click **Commit changes...**
2. In the "Commit message" field, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "[Creating a commit with multiple authors](https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors)."
3. If you have more than one email address associated with your account on GitHub.com, click the email address drop-down menu and select the email address to use as the Git author email address. Only verified email addresses appear in this drop-down menu. If you enabled email address privacy, then a no-reply will be the default commit author email address. For more information about the exact form the no-reply email address can take, see "[Setting your commit email address](https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address)."



1. Below the commit message fields, decide whether to add your commit to the current branch or to a new branch. If your current branch is the default branch, you should choose to create a new branch for your commit and then create a pull request. For more information, see "[Creating a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request)."

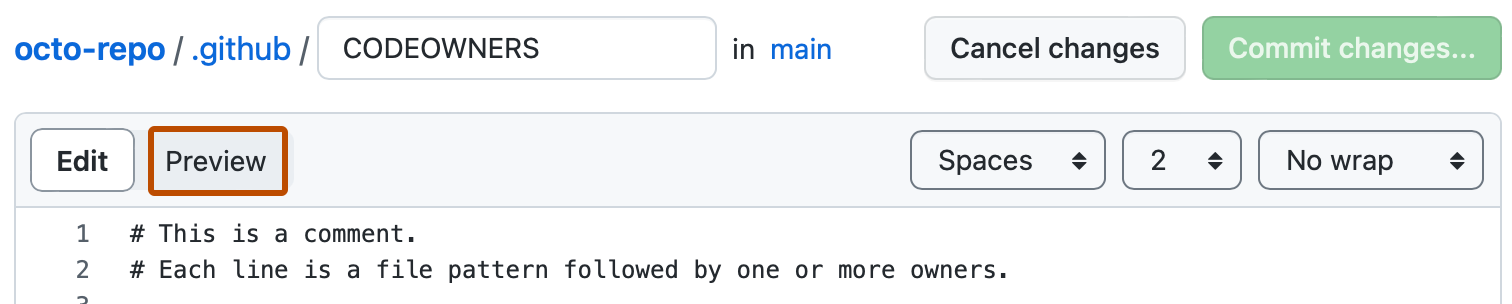


1. Click **Commit changes** or **Propose changes**.

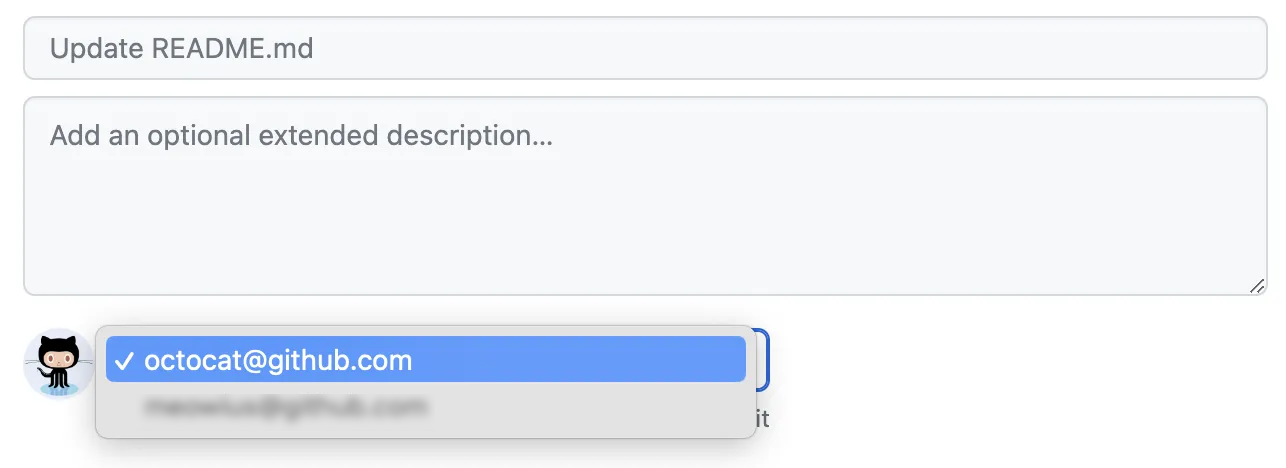
Editing files in another user's repository

When you edit a file in another user's repository, we'll automatically [fork the repository](https://docs.github.com/en/get-started/quickstart/fork-a-repo) and [open a pull request](https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/creating-a-pull-request) for you.

1. In another user's repository, browse to the folder that contains the file you want to edit. Click the name of the file you want to edit.
2. Above the file content, click . At this point, GitHub forks the repository for you.
3. In the text box, make any changes you need to the file.
4. Above the new content, click **Preview**.



1. Click **Commit changes...**
2. In the "Commit message" field, type a short, meaningful commit message that describes the change you made to the file. You can attribute the commit to more than one author in the commit message. For more information, see "[Creating a commit with multiple authors](https://docs.github.com/en/pull-requests/committing-changes-to-your-project/creating-and-editing-commits/creating-a-commit-with-multiple-authors)."
3. If you have more than one email address associated with your account on GitHub.com, click the email address drop-down menu and select the email address to use as the Git author email address. Only verified email addresses appear in this drop-down menu. If you enabled email address privacy, then a no-reply will be the default commit author email address. For more information about the exact form the no-reply email address can take, see "[Setting your commit email address](https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-personal-account-on-github/managing-email-preferences/setting-your-commit-email-address)."



1. Click **Propose changes**.
2. Type a title and description for your pull request.
3. Click **Create pull request**.

# Downloading source code archives

Overview of source code archives

You can download a snapshot of any branch, tag, or specific commit from GitHub.com. These snapshots are generated by the git archive command in one of two formats: tarball or zipball. Snapshots don't contain the entire repository history. If you want the entire history, you can clone the repository. For more information, see "Cloning a repository."

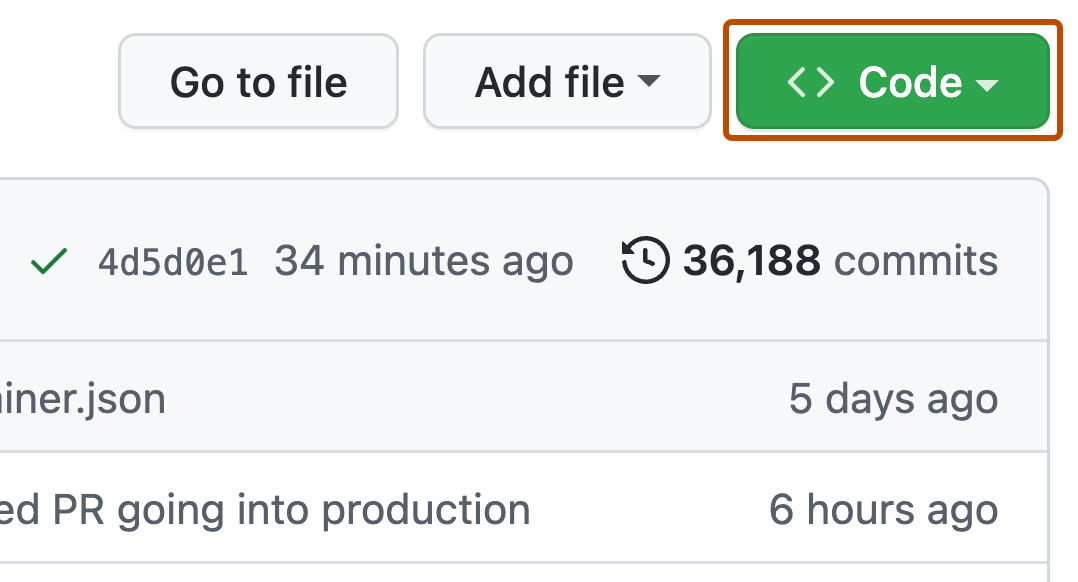
Downloading source code archives

You can download the source code archives in three ways.

Downloading source code archives from the repository view

On GitHub.com, navigate to the main page of the repository.

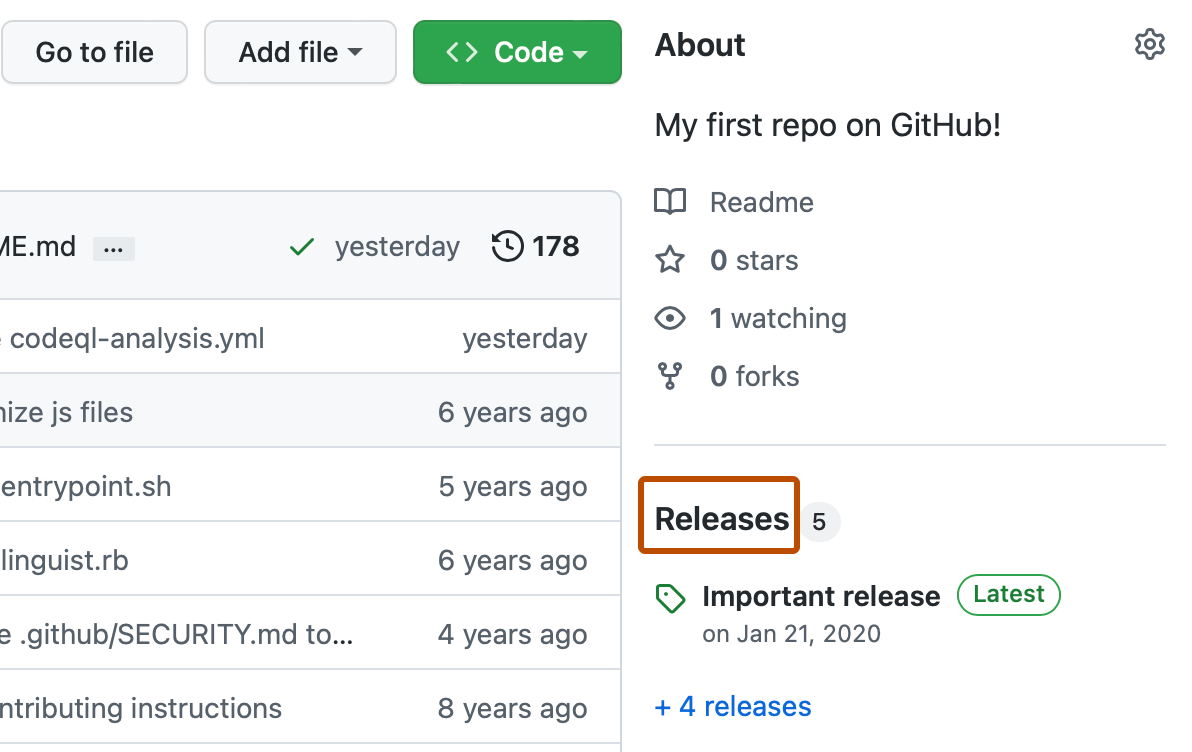
Above the list of files, click<> Code.



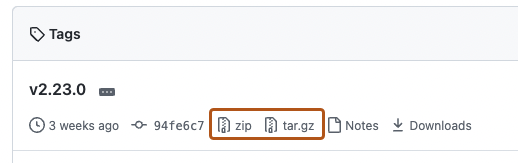
1. Click Download **ZIP**.

### [Downloading source code archives from a release](https://docs.github.com/en/repositories/working-with-files/using-files/downloading-source-code-archives" \l "downloading-source-code-archives-from-a-release)

1. On GitHub.com, navigate to the main page of the repository.
2. To the right of the list of files, click **Releases**.



1. At the top of the Releases page, click **Tags**.
2. To download the source code, click zip or  **tar.gz**.



## [Source code archive URLs](https://docs.github.com/en/repositories/working-with-files/using-files/downloading-source-code-archives" \l "source-code-archive-urls)

Source code archives are available at specific URLs for each repository. For example, consider the repository github/codeql. There are different URLs for downloading a branch, a tag, or a specific commit ID.

| Type of archive | Example | URL |
| --- | --- | --- |
| Branch | main | [https://github.com/github/codeql/archive/refs/](https://github.com/github/codeql/archive/refs/heads/main.tar.gz)**[heads/main](https://github.com/github/codeql/archive/refs/heads/main.tar.gz)**[.tar.gz](https://github.com/github/codeql/archive/refs/heads/main.tar.gz) |
| Tag | codeql-cli/v2.12.0 | [https://github.com/github/codeql/archive/refs/](https://github.com/github/codeql/archive/refs/tags/codeql-cli/v2.12.0.zip)**[tags/codeql-cli/v2.12.0](https://github.com/github/codeql/archive/refs/tags/codeql-cli/v2.12.0.zip)**[.zip](https://github.com/github/codeql/archive/refs/tags/codeql-cli/v2.12.0.zip) |
| Commit | aef66c4 | [https://github.com/github/codeql/archive/](https://github.com/github/codeql/archive/aef66c462abe817e33aad91d97aa782a1e2ad2c7.zip)**[aef66c462abe817e33aad91d97aa782a1e2ad2c7](https://github.com/github/codeql/archive/aef66c462abe817e33aad91d97aa782a1e2ad2c7.zip)**[.zip](https://github.com/github/codeql/archive/aef66c462abe817e33aad91d97aa782a1e2ad2c7.zip) |

**Note**: You can use either .zip or .tar.gz in the URLs above to request a zipball or tarball respectively.

## [Stability of source code archives](https://docs.github.com/en/repositories/working-with-files/using-files/downloading-source-code-archives" \l "stability-of-source-code-archives)

Source code archives are generated on request, cached for a while, and then deleted. If the same archive is requested again in the future, it'll be regenerated. It's important to understand what guarantees GitHub makes about source code archives.

* An archive of a commit ID will always have the same file contents whenever it's requested, assuming the commit ID is still in the repository and the repository's name has not changed.
* Because branches and tags can move to different commit IDs, future downloads of an archive may have different contents than previously downloaded archives of the same branch or tag. Assuming the branch or tag still points at the same commit ID, it will have the same file contents.
* The exact compression settings used to generate a zipball or tarball may change over time. The extracted contents won't change if the branch or tag doesn't change, but the outer compressed archive may have a different byte layout. GitHub will give at least six months' notice before changing compression settings.
* The name of the repository is part of the directory structure inside the archive. Therefore, if the repository name changes, the root directory name will change as well.

If you rely on stability of source code archives for reproducibility (ensuring you always get identical files inside the archive), we recommend using the [archives REST API](https://docs.github.com/en/rest/repos/contents" \l "download-a-repository-archive-tar) with a commit ID for :ref. Using the commit ID ensures you'll always get the same file contents inside the archive and you’ll be immune to repositories rewriting tags or moving branch heads.

If you rely on stability of archives for security (for example: to ensure you don't attempt to unzip a maliciously-crafted file), we recommend using releases instead of using source downloads. For more information, see "[About releases](https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases)."

You can use something like [this third-party GitHub action](https://github.com/softprops/action-gh-release) to create and push these files as part of your release process. The [Release Assets REST API](https://docs.github.com/en/rest/releases/assets" \l "get-a-release-asset) can later be used to retrieve them.

Press alt+up to activate

Bottom of Form

**Practical Related Questions:**

Write down steps to edit file from repository?

What do you mean by archive file?

How to download source code?