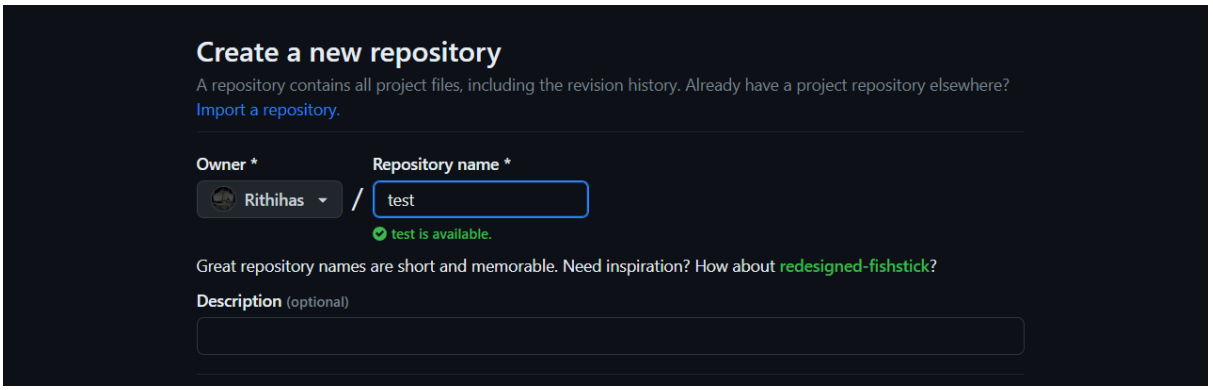
WE LAB

1.Create a git repository and clone it for changes and publish the changes using gitbash(Git commands)

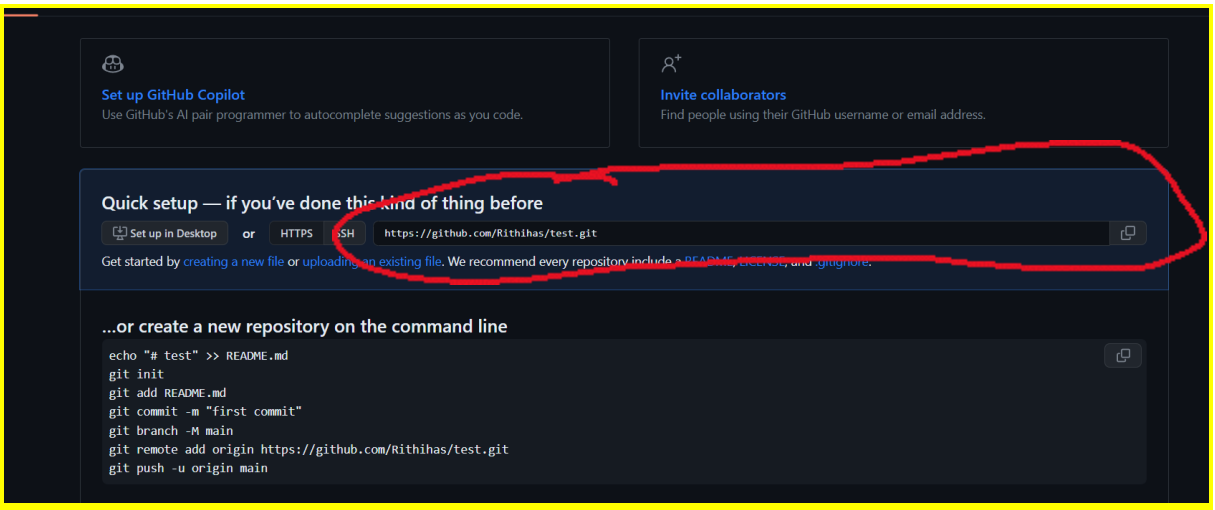
A) step 1:

Create a new repository.



Step 2:

Copy the repository link.



Step 3:

Create an empty folder on your desktop and open gitbash / command prompt in that folder. (navigate to that folder using cd command).

Step 4:

Clone the repository into the current folder using git clone.

```
D:\CVRnotes\year3sem2\WE\githubtest>git clone https://github.com/Rithihas/test.git
Cloning into 'test'...
warning: You appear to have cloned an empty repository.

D:\CVRnotes\year3sem2\WE\githubtest>
```

Step 5:

Create any file and save it in the cloned folder.

Step 6:

Use “ git add . “ to add the file to the staging area. And then use git commit -m “message” to commit changes.

```
D:\CVRnotes\year3sem2\WE\githubtest\test>git add .

D:\CVRnotes\year3sem2\WE\githubtest\test>git commit -m "created file"
[main (root-commit) d217ff5] created file
 1 file changed, 1 insertion(+)
 create mode 100644 testfile.txt
```

Step 7:

push changes to github using git push.

```
D:\CVRnotes\year3sem2\WE\githubtest\test>git push
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 225 bytes | 225.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/Rithihas/test.git
 * [new branch]      main -> main
```

Note:

If it asks for credentials try:

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

$git config --global user.email “youremail@gmail.com”

$git config --global user.password "yourpassword"

2. Working of ES6 features like arrow functions, destructuring and function generators.

Arrow functions :

```
// Basic arrow function
const greet = () => {
  console.log("Hello, world!");
};

greet(); // Output: Hello, world!

// Arrow function with parameters
const sum = (a, b) => {
  return a + b;
};

console.log(sum(2, 3)); // Output: 5

// Arrow function with implicit return
const multiply = (a, b) => a * b;

console.log(multiply(4, 5)); // Output: 20

// Arrow function with a single parameter
const square = x => x * x;

console.log(square(3)); // Output: 9
```

destructuring:

```
// Destructuring arrays
const numbers = [1, 2, 3, 4, 5];

const [first, second, ...rest] = numbers;

console.log(first); // Output: 1
console.log(second); // Output: 2
console.log(rest); // Output: [3, 4, 5]


// Destructuring objects
const person = {
  name: "John",
  age: 30,
  address: {
    city: "New York",
    country: "USA",
  },
};

const { name:vape, age: agro, address: { city, country } } = person;
```

```
console.log(vape); // Output: John
console.log("age is : ",agro); // Output: 30
console.log(city); // Output: New York
console.log(country); // Output: USA
```

generator functions:

```
// Generator function
function* numberGenerator() {
    yield 1;
    yield 2;
    yield 3;
    yield 4;
    yield 5;
  }

  // Create an instance of the generator
  const generator = numberGenerator();

  // Iterate over the values using the generator
  console.log(generator.next().value); // Output: 1
  console.log(generator.next().value); // Output: 2
  console.log(generator.next().value); // Output: 3
  console.log(generator.next().value); // Output: 4
  console.log(generator.next().value); // Output: 5
  console.log(generator.next().value); // Output: undefined (no more yield
values)

  // Generator function with parameters
  function* rangeGenerator(start, end, step) {
    for (let i = start; i <= end; i += step) {
      yield i;
    }
  }

  const range = rangeGenerator(1, 10, 2);

  // Iterate over the range using the generator
  console.log(range.next().value); // Output: 1
  console.log(range.next().value); // Output: 3
  console.log(range.next().value); // Output: 5
  console.log(range.next().value); // Output: 7
  console.log(range.next().value); // Output: 9
  console.log(range.next().value); // Output: undefined (no more yield values)
```

3. Explain the node modules: os,http,fs etc

1. os module:

```
var os = require('os');

console.log('cpu architecture: '+os.arch());
```

```
console.log('free memory :'+os.freemem());

console.log('total memory: '+os.totalmem());

console.log('os type : ' + os.type());
```

2. http module:

```
var http = require('http');

http.createServer(function(req,res){
    res.write('hello world!');
    res.end();
}).listen(8070);
```

3. fs module:

```
const fs = require('fs');

// Read from a file
fs.readFile('input.txt', 'utf8', (err, data) => {
  if (err) {
    console.error(err);
    return;
  }

  console.log('File content:');
  console.log(data);

  // Write to a file
  const content = data.toUpperCase();
  fs.writeFile('output.txt', content, 'utf8', (err) => {
    if (err) {
      console.error(err);
      return;
    }

    console.log('Data has been written to the file successfully.');
  });
});
```

4. typescript classes.

Installation:

1.”npm install -g typescript”

2. “Set-ExecutionPolicy -Scope CurrentUser “ (executionpolicy value is 1)

3. create file with .ts extension.

4. “tsc filename.ts “ to compile to js file.

5. run the js file using “node filename.js”

```
// Define a class
class Animal {
    private name: string;
    private age: number;

    constructor(name: string, age: number) {
      this.name = name;
      this.age = age;
    }

    public introduce(): void {
      console.log(`Hi, my name is ${this.name} and I am ${this.age} years
old.`);
    }
  }

  // Create instances of the class
  const lion = new Animal("Simba", 5);
  const elephant = new Animal("Dumbo", 10);

  // Call class methods
  lion.introduce();     // Output: Hi, my name is Simba and I am 5 years old.
  elephant.introduce(); // Output: Hi, my name is Dumbo and I am 10 years old.
```

5. typescript generics.

```
// Generic class
class Box<T> {
    private item: T;

    constructor(item: T) {
      this.item = item;
    }

    public getItem(): T {
      return this.item;
    }
  }

  // Create instances of the generic class
  const box1 = new Box<number>(10);
  console.log(box1.getItem()); // Output: 10

  const box2 = new Box<string>("Hello");
  console.log(box2.getItem()); // Output: Hello
```

```typescript
// Generic function
function printArray<T>(array: T[]): void {
  for (let item of array) {
    console.log(item);
  }
}

// Call the generic function
const numbers: number[] = [1, 2, 3, 4, 5];
printArray<number>(numbers); // Output: 1 2 3 4 5

const names: string[] = ["Alice", "Bob", "Charlie"];
printArray<string>(names); // Output: Alice Bob Charlie
```

[redacted]

Remove everything in index.js file and type this:

```typescript
import React from 'react';
import ReactDOM from 'react-dom';

// functional JSX component
const MyComponent = () => {
  const name = 'John Doe';
  const age = 30;

  return (
    <div>
      <h1>Hello, using functional component : {name}!</h1>
      <p>You are {age} years old.</p>
    </div>
  );
};

class MyComponent2 extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      name: 'John Doe',
      age: 30
    };
  }

  render() {
    const { name, age } = this.state;

    return (
      <div>
```