1. JDK : Java 18 [Not JRE, we will require JDK]

2. Windows OS?

https://www.oracle.com/java/technologies/javase/jdk18-archive-downloads.html

3. NodeJS : https://nodejs.org/en [v20.x]

NPM : Node Package Manager

https://www.npmjs.com/

RedHat : RPM

Ubuntu : apt or apt-get

NodeJS : npm

4. MySQL Community Server : [v8.0.36] - Mysql installer MSI

Dual License Policy :

a. Community Version == Community Edition [Free and open source]

b. Enterprise version == Paid version

https://dev.mysql.com/downloads/mysql/ - Windows (x86, 32-bit), MSI Installer [285 MB - download]

5. MySQL --> Indexes in MYSQL?

Databases ==> Data Stores

- SQL DB

- NoSQL DB

- In-memory cache implementation

- In-memory storages

Traditional env : HDD/SDD

new age of computation : SSD/HDD --> Memory

Memory vs HDD Mem : is about 8-9X faster

Faster : RW operations

AWS : Redis

In-memory storages :: Ephemeral Storages

Data state | Data storage :

- Persistent storage : Even when the machine [storage device/server/database server/etc] is switched OFF : data remains

- Non-persistent stotage : Even when the machine [storage device/server/database server/etc] is switched OFF : data goes away : Ephemeral

Data state is NON-Persistent ?

Enterprise Apps :

- Speed

- Performance

- Scalability

6. ReactJS :

https://react.dev/learn/installation

NodeJS + NPM Installed

[NPM CRA - Create-React-App] npm install -g create-react-app

npm install --save react react-dom

npx create-react-app <name of the app>

NPM vs NPX

7. JUnit & TestNg

Unit + Integration:

TESTNG : https://testng.org/

Unit testing :

JUnit : Add the dependencies

Java - Junit

.NET/C# - NUnit

Python - PyUnit

-> Types of testing

- Unit testing

- Functional testing

- E2E [end-to-end testing]

- API Testing

- Security testing

- Peformance testing

- Monkey testing

- Fuzzy testing -> after you finished the above

Service project : 12-15 types of testing

Product project : 27-29 types of testing

POM.XML

Where is POM.XML coming from and what does it do?

pom.xml : project object model

https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

What is Maven : Build Tool

Enterprise Apps : Dev Stage - test stage - release stage - production stage[phases]

- Dev code

- Test code

- Artifacts

- Build tools : Java --> ANT/Maven/Gradle + .NET + NodeJS apps --> npm

CI/CD Process : DevOps cycle

Jenkins :

CI : Continuous Integration

CD : Continuous Delivery/Deployment

- Release tools

- deployment tools

8. Selenium : Java + Web Driver

9. JMeter

https://jmeter.apache.org/download\_jmeter.cgi

Enterprise Apps :

a. What is performance engineering [web vs mobile app vs Cloud Native Apps]

b. What is a different performance vs scaling

c. What is the concept horizontal scaling vs vertical scaling

d. Sample app : LinkedIn :

S1 : Currently has 1000 services --[new features]--> 1200 services [in terms of scaling : what to scale : Horizontal OR Vertical Scaling]

S2 : Currently has 1000000 end-users --[new features]--> 2000000 end-users [in terms of scaling : what to scale : Horizontal OR Vertical Scaling]

Project Design :

- Computational Power [Algorithm : Small thread vs Long thread]

- Transactional Processing : RW Activities [eg banking app : Create/Insert vs Select queries]

- Storage capacity : Instagram --> reels stored/comments/profiles stored || Accounts [active/non-active]

10. Github + Git

- Local windows : SCM-GIT

https://git-scm.com/downloads

- Github account : https://github.com/

a. Create your Github account

b. Create a repo

Each of you :

Repo1 - practice

repo2 - assignments

repo3 - project repo [later]

[8:43 PM] Dibya (Guest)

https://www.w3schools.com/git/

[8:44 PM] Dibya (Guest)

https://www.youtube.com/watch?v=RGOj5yH7evk

[8:44 PM] Dibya (Guest)

https://www.datacamp.com/tutorial/github-and-git-tutorial-for-beginners

Git/Github concepts :

a. SCM - Software Configuration management

b. Git architecture

c. Git commands : Branch/Push/Pull/Merge/Clone/Rebase

POSTMAN :

https://www.postman.com/downloads/

----------------------------------------------------------------------------------------------------------------------------------------------

What is use of indexes in MYSql

In MySQL, an index is a database object that improves the speed of data retrieval operations on a table.

It works similarly to an index in a book, helping to quickly locate specific information without having to scan the entire content.

Indexes in MySQL serve several purposes:

Improving Query Performance:

Indexes provide a faster way to locate rows in a table, especially when filtering or sorting data based on certain columns.

This can significantly improve the performance of SELECT, WHERE, and ORDER BY clauses in queries.

----------------------------------------------------------------------------------------------------------------------------------------------

Database Types:

a. Relational Databases:

Organize data into tables with predefined relationships.

Examples: MySQL, PostgreSQL, Oracle, Microsoft SQL Server.

b. NoSQL Databases:

Do not use a traditional tabular relational database model.

Types include document stores, key-value stores, column-family stores, and graph databases.

Examples: MongoDB (document store), Redis (key-value store), Cassandra (column-family store), Neo4j (graph database).

In-memory cache implementation refers to the use of an in-memory data store to temporarily store and quickly

retrieve frequently accessed or computationally expensive data.

----------------------------------------------------------------------------------------------------------------------------------------------

Types of Testing :

Functional Testing :

UnitTesting : Unit testing is a method of testing individual units or components of a software application.

It is typically done by developers and is used to ensure that the individual units of the software are working as intended.

Unit tests are usually automated and are designed to test specific parts of the code, such as a particular function or method.

Unit testing is done at the lowest level of the software development process, where individual units of code are tested in isolation.

Advanteges of Unit testing :

It helps to identify bugs early in the development process before they become more difficult and expensive to fix.

It helps to ensure that changes to the code do not introduce new bugs.

It makes the code more modular and easier to understand and maintain.

It helps to improve the overall quality and reliability of the software.

Integration testing :

Integration testing is a method of testing how different units or components of a software application interact with each other.

It is used to identify and resolve any issues that may arise when different units of the software are combined.

Integration testing is typically done after unit testing and before functional testing and is used to verify that

the different units of the software work together as intended.

Advanteges of Integration testing :

It helps to improve the overall reliability and stability of the software.

It’s important to keep in mind that Integration testing is essential for complex systems where different components are integrated together.

(a) Black Box testing:- It is used for validation. In this, we ignore internal working mechanisms and focus on what is the output?.

(b) White box testing:- It is used for verification. In this, we focus on internal mechanisms i.e. how the output is achieved?.

E2E Testing :

End-to-end (E2E) testing is a software testing methodology that verifies the working order of a software product

in a start-to-finish process. End-to-end testing verifies that all components of a system can run under real-world scenarios.

There are two types :

vertical testing --> This refers to thetesting in layers Ex: White box testing

Horizental testing --> Example :BlackBox Testing

API Testing :

API testing is a type of software testing that involves testing application programming interfaces (APIs) directly

and as part of integration testing to determine if they meet expectations for functionality,

reliability, performance, and security. Since APIs lack a GUI, API testing is performed at the message layer.

Performance Testing :

Performance testing is a testing measure that evaluates the speed, responsiveness and stability of a computer, network,

software program or device under a workload. Organizations will run performance tests to identify performance-related bottlenecks.

Security Testing :

Security testing is an important aspect of software testing focused on identifying and addressing security vulnerabilities in a software application.

It aims to ensure that the software is secure from malicious attacks, unauthorized access, and data breaches.

Monkey Testing :

Monkey testing is a type of software testing in which a software or application is tested using random inputs with the

sole purpose of trying and breaking the system. There are no rules in this type of testing.

It completely works on the tester's mood or gut feeling and experience.

--------------------------------------------------------------------------------------------------------------------------------------------------

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

To check the version : git --version

git version 2.30.2.windows.1

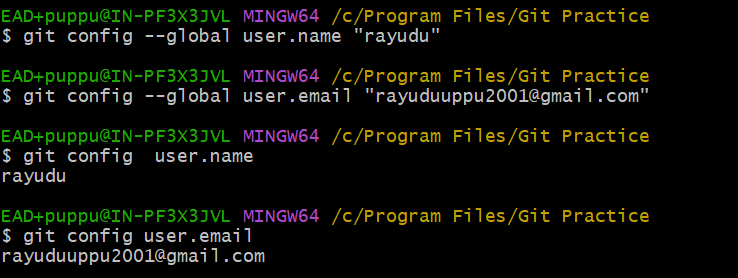
Configure Git : git config --global user.name "w3schools-test"

git config --global user.email "test@w3schools.com"

Creating Git Folder :

mkdir myproject

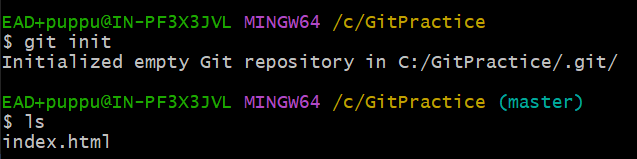
cd myproject



**Initialize Git :**

git init

Initialized empty Git repository in /Users/user/myproject/.git/



Master ---🡪 is the main repository

Create a file : Index.html

Files in your Git repository folder can be in one of 2 states:

1. Tracked - files that Git knows about and are added to the repository
2. Untracked - files that are in your working directory, but not added to the repository

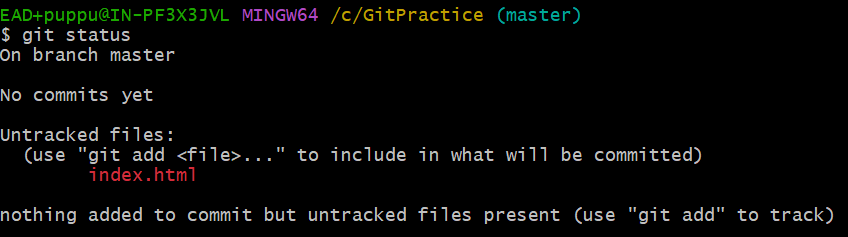
Git Staging Environment

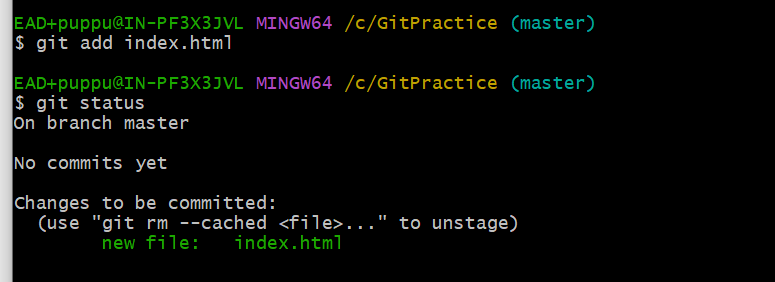
One of the core functions of Git is the concepts of the Staging Environment, and the Commit.

As you are working, you may be adding, editing and removing files. But whenever you hit a milestone or finish a part of the work, you should add the files to a Staging Environment.

**Staged** files are files that are ready to be **committed** to the repository you are working on. You will learn more about commit shortly.

1. For now, we are done working with index.html. So we can add it to the Staging Environment:





Git Add More than One File

You can also stage more than one file at a time. Let's add 2 more files to our working folder. Use the text editor again.

A README.md file that describes the repository

**README.md**

# hello-world  
Hello World repository for Git tutorial  
This is an example repository for the Git tutoial on https://www.dxc.com  
  
This repository is built step by step in the tutorial.

**bluestyle.css :**

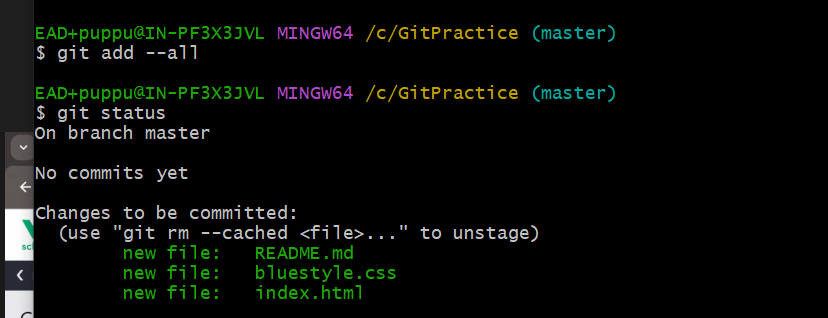
body {  
background-color: lightblue;  
}  
  
h1 {  
color: navy;  
margin-left: 20px;  
}

**index.html:**

<!DOCTYPE html>  
<html>  
<head>  
<title>Hello World!</title>  
<link rel="stylesheet" href="bluestyle.css">  
</head>  
<body>  
  
<h1>Hello world!</h1>  
<p>This is the first file in my new Git Repo.</p>  
  
</body>  
</html>

Now add all files in the current directory to the Staging Environment:

git add --all



Note: The shorthand command for git add --all is git add -A

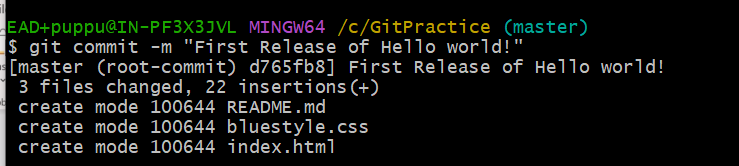
Git Commit:

Since we have finished our work, we are ready move from stage to commit for our repo.

Adding commits keep track of our progress and changes as we work. Git considers each commit change point or "save point". It is a point in the project you can go back to if you find a bug, or want to make a change.

When we commit, we should always include a message.

By adding clear messages to each commit, it is easy for yourself (and others) to see what has changed and when.

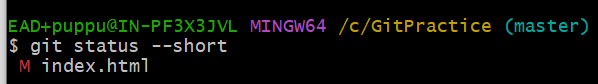


Git Commit without Stage

Sometimes, when you make small changes, using the staging environment seems like a waste of time. It is possible to commit changes directly, skipping the staging environment. The -a option will automatically stage every changed, already tracked file.

Index.html add new line :

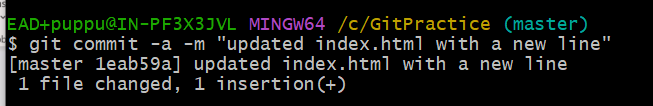
<p>This is the first file in my new Git Repo.</p>



**Note:** Short status flags are:

1. ?? - Untracked files
2. A - Files added to stage
3. M - Modified files
4. D - Deleted files

we see the file we expected is modified. So let's commit it directly:



Git Commit Log

To view the history of commits for a repository, you can use the log command:



Git Help

If you are having trouble remembering commands or options for commands, you can use Git help.

There are a couple of different ways you can use the help command in command line:

1. git *command* -help -  See all the available options for the specific command
2. git help --all -  See all possible commands

New Git Branch

Let add some new features to our index.html page.

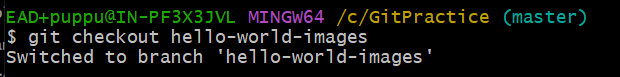
We are working in our local repository, and we do not want to disturb or possibly wreck the main project.

So we create a new branch:



We can see the new branch with the name "hello-world-images", but the \* beside master specifies that we are currently on that branch.

checkout is the command used to check out a branch. Moving us from the current branch, to the one specified at the end of the command:



Now we have moved our current workspace from the master branch, to the new branch

Open your favourite editor and make some changes.

For this example, we added an image (img\_hello\_world.jpg) to the working folder and a line of code in the index.html file:

Index.html :

<!DOCTYPE html>

<html>

<head>

<title>Hello World!</title>

<link rel="stylesheet" href="bluestyle.css">

</head>

<body>

<h1>Hello world!</h1>

<div><img src="C:\Users\ypavani\OneDrive - DXC Production\Pictures\DXC Logo\DXC Logo 2C Horiz\_White RGB.png" alt="DXC LOGO Bangalore"

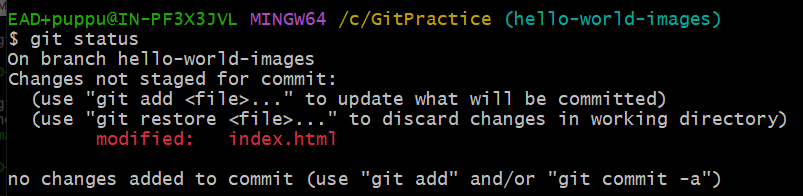
style="width:100%;max-width:960px"></div>

<p>A new line in our file!</p>

<p>This is the first file in my new Git Repo.</p>

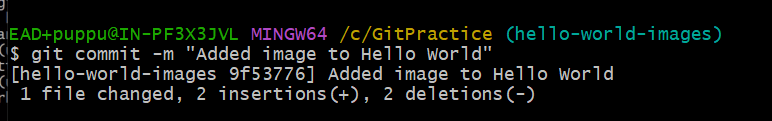
</body>

</html>



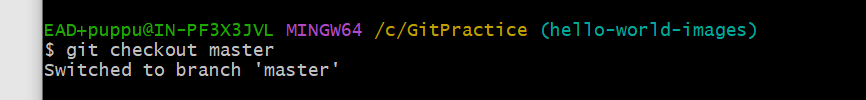
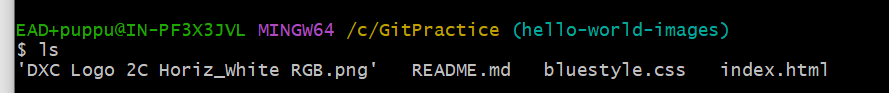


We are happy with our changes. So we will commit them to the branch:



Note: Using the -b option on checkout will create a new branch, and move to it, if it does not exist

Switching Between Branches

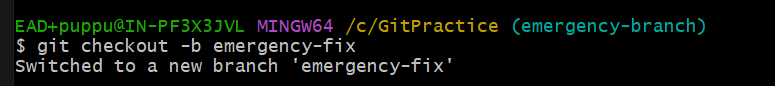


Emergency Branch

Now imagine that we are not yet done with hello-world-images, but we need to fix an error on master.

I don't want to mess with master directly, and I do not want to mess with hello-world-images, since it is not done yet.

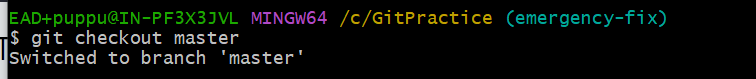
So we create a new branch to deal with the emergency:

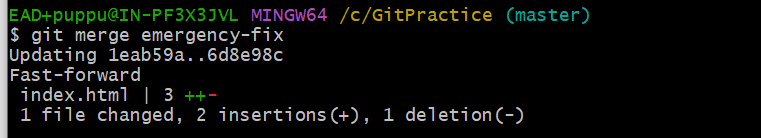


Merge Branches

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to change to the master branch:





## Merge Branches

We have the emergency fix ready, and so let's merge the master and emergency-fix branches.

First, we need to change to the master branch

Delete Branches :

## 

## Merge Conflict:

## first checkout to hello-world-image

## Add image to index.html

## Index.html:

## <!DOCTYPE html>

## <html>

## <head>

## <title>Hello World!</title>

## <link rel="stylesheet" href="bluestyle.css">

## </head>

## <body>

## <h1>Hello world!</h1>

## <div><img src="C:\GitPractice\DXC Logo 2C Horiz\_White RGB.png" alt="DXC TECHNOLOGY "

## style="width:100%;max-width:960px"></div>

## <p>This is the first file in my new Git Repo.</p>

## <p>A new line in our file!</p>

## <div><img src="C:\GitPractice\jerry.webp" alt="DXC" style="width:100%;max-width:640px"></div>

## </body>

## </html>

## And delete the hello-world-images branch:

## 

## Push Local Repository to GitHub

## Since we have already set up a local Git repo, we are going to push that to GitHub:

## 

## 

## Edit Code in GitHub

In addition to being a host for Git content, GitHub has a very good code editor.

Let's try to edit the README.md file in GitHub. Just click the edit button:

## 

## 

## Finally we edited the code

Pulling to Keep up-to-date with Changes

When working as a team on a project, it is important that everyone stays up to date.

Any time you start working on a project, you should get the most recent changes to your local copy.

With Git, you can do that with pull.

pull is a combination of 2 different commands:

* fetch
* merge

## Git Fetch

fetch gets all the change history of a tracked branch/repo.

So, on your local Git, fetch updates to see what has changed on GitHub:

## 

## 

## To see the difference between our local master and origin/master

## Git Merge

merge combines the current branch, with a specified branch.

We have confirmed that the updates are as expected, and we can merge our current branch (master) with origin/master:

## 

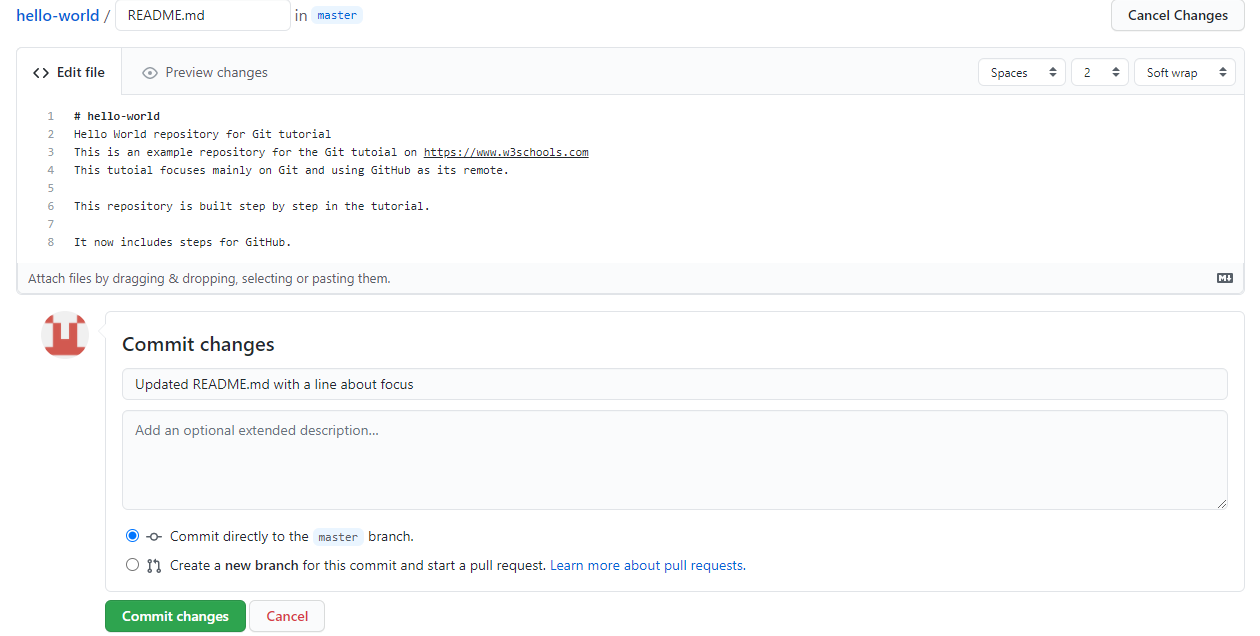
## 

## Git Pull

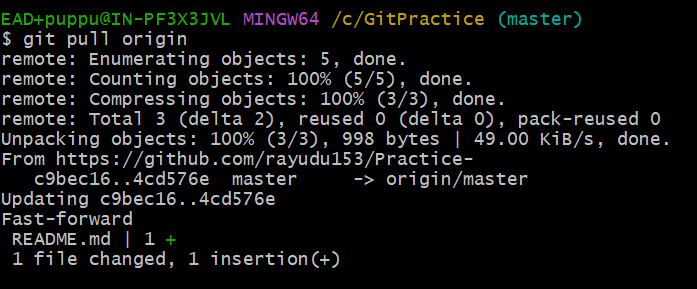
But what if you just want to update your local repository, without going through all those steps?

pull is a combination of fetch and merge. It is used to pull all changes from a remote repository into the branch you are working on.

Make another change to the Readme.md file on GitHub.



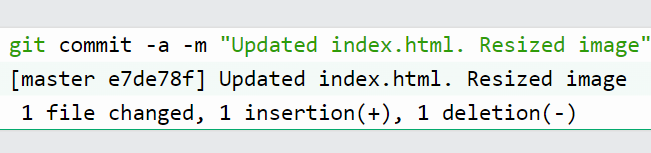
Use pull to update our local Git:



## Push Changes to GitHub

Let's try making some changes to our local git and pushing them to GitHub.

Index.html : <div><img src="img\_hello\_world.jpg" alt="Hello World from Space" style="width:100%;max-width:640px"></div>



## 

