ESEARCH INNOVATION DISCOVERY

RID BHARAT (समस्या का समाधान)

RUN BY TWKSAA WELFARE FOUNDATION

Reg.No: 048884
Foundation Day
05-09-2024

DevOps E-Book

Er. Rajesh Prasad(B.E, M.E)
Founder: RID Organization

- RID ORGANIZATION यानि Research, Innovation and Discovery संस्था जिसका मुख्य उदेश्य हैं आने वाले समय में सबसे पहले NEW (RID, PMS & TLR) की खोज, प्रकाशन एवं उपयोग भारत की इस पावन धरती से भारतीय संस्कृति, सभ्यता एवं भाषा में ही हो |
- देश, समाज, एवं लोगों की समस्याओं का समाधान NEW (RID, PMS & TLR) के माध्यम से किया जाये इसके लिए ही मैं राजेश प्रसाद इस RID संस्था की स्थपना किया हूँ|
- Research, Innovation & Discovery में रूचि रखने वाले आप सभी विधार्थियों, शिक्षकों एवं बुधीजिवियो से मैं आवाहनं करता हूँ की आप सभी इस RID संस्था से जुड़ें एवं अपने बुध्दि, विवेक एवं प्रतिभा से दुनियां को कुछ नई (RID, PMS & TLR) की खोजकर, बनाकर एवं अपनाकर लोगों की समस्याओं का समाधान करें |

त्वक्सा DevOps के इस ई-पुस्तक में आप DevOps से जुड़ी सभी बुनियादी अवधारणाएँ सीखेंगे। मुझे आशा है कि इस ई-पुस्तक को पढ़ने के बाद आपके ज्ञान में वृद्धि होगी और आपको कंप्यूटर विज्ञान के बारे में और अधिक जानने में रुचि होगी

"In this E-Book of TWKSAA DevOps you will learn all the basic concepts related to DevOps. I hope after reading this E-Book your knowledge will be improved and you will get more interest to know more thing about computer Science".

RID हमें क्यों करना चाहिए

(Research)

अनुसंधान हमें क्यों करना चाहिए ?

Why should we do research?

1. नई ज्ञान की प्राप्ति (Acquisition of new knowledge)
2. समस्याओं का समाधान(To Solving problems)
3. सामाजिक प्रगति (To Social progress)
4. विकास को बढ़ावा देने(To promote development)
5. तकनीकी और व्यापार में उन्नति(To advances in technology & business)
6. देश विज्ञान और प्रौद्योगिकी के विकास(To develop the country's science & technology)

(Innovation)

नवीनीकरण हमें क्यों करना चाहिए ?

Why should we do Innovation?

1. प्रगति के लिए(To progress)
2. परिवर्तन के लिए(For change)
3. उत्पादन में सुधार(To Improvement in production)
4. समाज को लाभ(To Benefit to society)
5. प्रतिस्पर्धा में अग्रणी (To be ahead of competition)
6. देश विज्ञान और प्रौद्योगिकी के विकास।
(To develop the country's science & technology)

(Discovery)

खोज हमें क्यों करना चाहिए?

Why should we do Discovery?

1. नए ज्ञान की प्राप्ति(Acquisition of new knowledge)
2. अविष्कारों की खोज(To Discovery of inventions)
3. समस्याओं का समाधान(To Solving problems)
4. ज्ञान के विकास में योगदान(Contribution to development of knowledge)
5. समाज के उन्नति के लिए (for progress of society)
6. देश विज्ञान और तकनीक के विकास
(To develop the country's science & technology)

New Technology पर Research करने के लिए सम्पर्क करें: ridorg.in@gmail.com Mob.No: 9202707903

❖ Research(अनुसंधान):

- अनुसंधान एक प्रणालीकरण कार्य होता है जिसमें विशेष विषय या विषय की नई ज्ञान एवं समझ को प्राप्त करने के लिए सिद्धांतिक जांच और अध्ययन किया जाता है। इसकी प्रक्रिया में डेटा का संग्रह और विश्लेषण, निष्कर्ष निकालना और विशेष क्षेत्र में मौजूदा ज्ञान में योगदान किया जाता है। अनुसंधान के माध्यम से विज्ञान, प्रोधोगिकी, चिकित्सा, सामाजिक विज्ञान, मानविकी, और अन्य क्षेत्रों में विकास किया जाता है। अनुसंधान की प्रक्रिया में अनुसंधान प्रश्न या कल्पनाएँ तैयार की जाती हैं, एक अनुसंधान योजना डिज़ाइन की जाती है, डेटा का संग्रह किया जाता है, विश्लेषण किया जाता है, निष्कर्ष निकाला जाता है और परिणामों को उचित दर्शाने के लिए समाप्ति तक पहुंचाया जाता है।

❖ Innovation(नवीनीकरण): -

- Innovation एक विशेषता या नई विचारधारा की उत्पत्ति या नवीनीकरण है। यह नए और आधुनिक विचारों, तकनीकों, उत्पादों, प्रक्रियाओं, सेवाओं या संगठनात्मक ढंगों का सृजन करने की प्रक्रिया है जिससे समस्याओं का समाधान, प्रतिस्पर्धा में अग्रणी होने, और उपयोगकर्ताओं के अनुकूलता में सुधार किया जा सकता है।

❖ Discovery (आविष्कार):

- Discovery का अर्थ होता है "खोज" या "आविष्कार"। यह एक विशेषता है जो किसी नए ज्ञान, अविष्कार, या तत्व की खोज करने की प्रक्रिया को संदर्भित करता है। खोज विज्ञान, इतिहास, भूगोल, तकनीक, या किसी अन्य क्षेत्र में हो सकती है। इस प्रक्रिया में, व्यक्ति या समूह नए और अज्ञात ज्ञान को खोजकर समझने का प्रयास करते हैं और इससे मानव सभ्यता और विज्ञान-तकनीकी के विकास में योगदान देते हैं।

नोट : अनुसंधान विशेषता या विषय पर नई ज्ञान के प्राप्ति के लिए सिस्टमैटिक अध्ययन है, जबकि आविष्कार नए और अज्ञात ज्ञान की खोज है।

	सुविचार:
1.	**समस्याओं का समाधान करने का उतम मार्ग हैं |** → शिक्षा ,RID, प्रतिभा, सहयोग, एकता एवं समाजिक-कार्य
2.	**एक इंसान के लिए जरूरी हैं |** → रोटी, कपड़ा, मकान, शिक्षा, रोजगार, इज्जत और सम्मान
3.	**एक देश के लिए जरूरी हैं |** - → संस्कृति-सभ्यता, भाषा, एकता, आजादी, संविधान एवं अखंडता
4.	**सफलता पाने के लिए होना चाहिए |** → लक्ष्य, त्याग, इच्छा-शक्ति, प्रतिबद्धता, प्रतिभा, एवं सतता
5.	**मरने के बाद इंसान छोडकर जाता हैं |** → शरीर, अन-धन, घर-परिवार, नाम, कर्म एवं विचार
6.	**मरने के बाद इंसान को इस धरती पर याद किया जाता हैं उनके**
	➔ नाम, काम, दान, विचार, सेवा-समपर्ण एवं कर्मो से...

आशीर्वाद (बड़े भैया जी)

Mr. RAMASHANKAR KUMAR

मार्गदर्शन एवं सहयोग

Mr. GAUTAM KUMAR

.सोच है जिनकी नई कुछ कर दिखाने की, खोज है रीड संस्था को उन सभी इंसानों की.

"अगर आप भी **Research, Innovation and Discovery** के क्षेत्र में रूचि रखतें हैं एवं अपनी प्रतिभा से दुनियां को कुछ नया देना चाहतें एवं अपनी समस्या का समाधान **RID** के माध्यम से करना चाहतें हैं तो **RID ORGANIZATION** (रीड संस्था) से जरुर जुड़ें" ‖ धन्यवाद ‖ **Er. Rajesh Prasad (B.E, M.E)**

CONTENTS

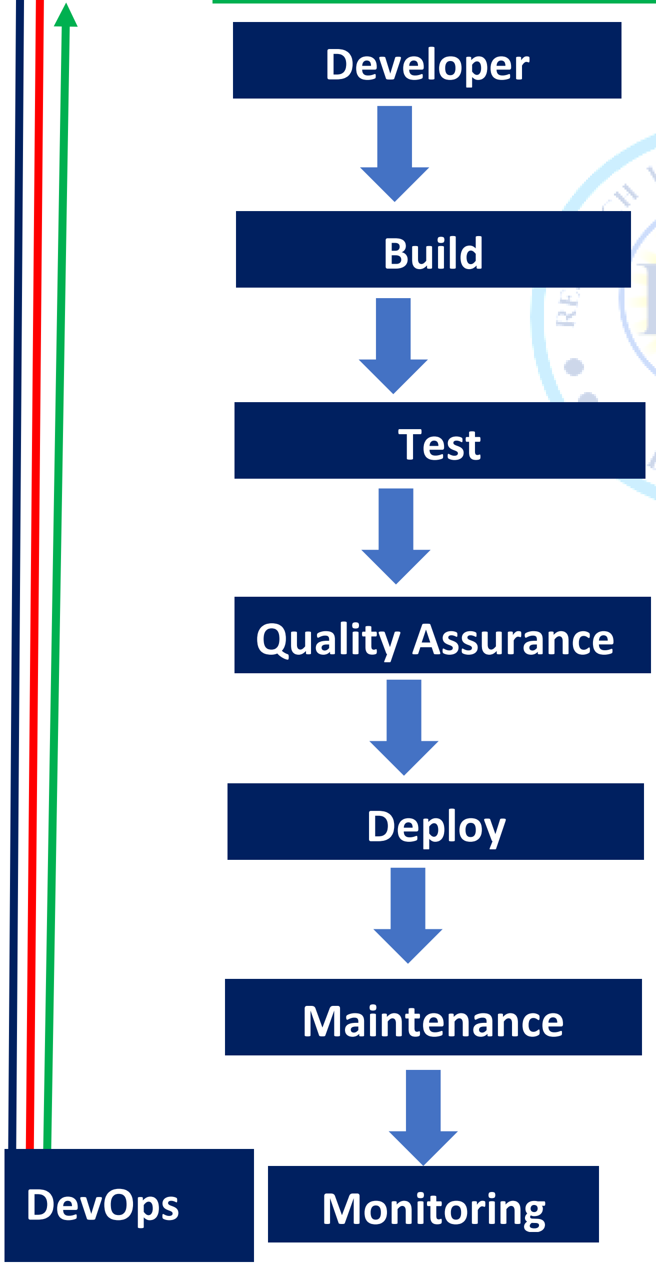
DevOps: - the term DevOps is a combination of two words i.e. Development and operations.

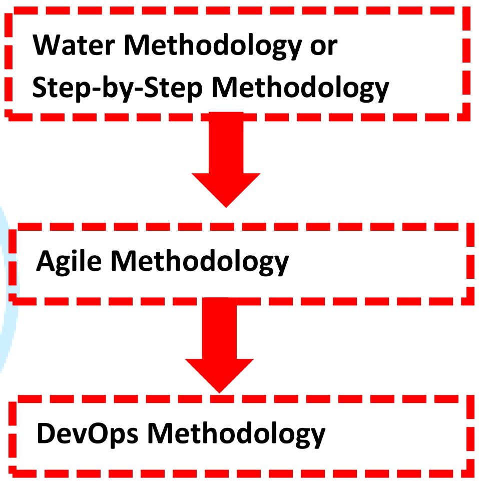
- DevOps is a Methodology that allows a single team to manage the entire application development life cycle. The development, testing, deployment and operations.
- Object of DevOps is to shorten the system's development life cycle.
- DevOps is a software development approach through which superior quality software can be developed quickly and with more reliability.

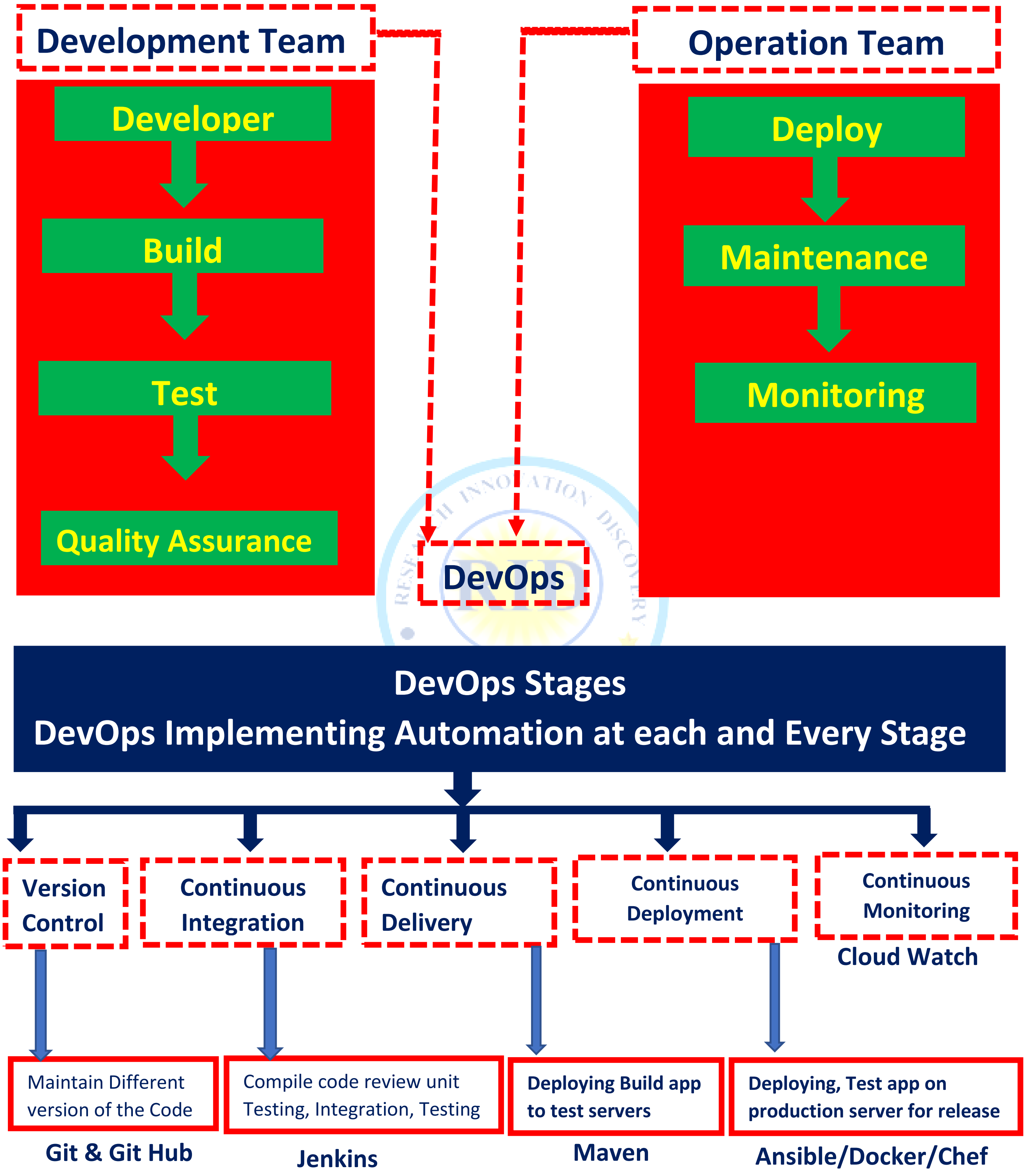
Why Organization Needs DevOps Engineer.

1. Fast Delivery
2. Higher Quality
3. Less Capex+Opex
4. Reduced Outages
5. High Availability
6. Scabble, Flexible & Reliable

SDCL: - Software Development Life Cycle.






Development Team
Developer
Build
Test
Quality Assurance
Operation Team
Deploy
Maintenance
Monitoring
DevOps
DevOps Stages
DevOps Implementing Automation at each and Every Stage
Version Control
Continuous Integration
Continuous Delivery
Continuous Deployment
Continuous Monitoring
Cloud Watch
Maintain Different version of the Code
Compile code review unit Testing, Integration, Testing
Deploying Build app to test servers
Deploying, Test app on production server for release
Git & Git Hub
Jenkins
Maven
Ansible/Docker/Chef

DevOps Architecture Features

1. Automation
2. Collaboration
3. Integration
4. Configuration Management

DevOps Life Cycle

1. Continuous Development
2. Continuous Integration
3. Continuous Deployment
4. Continuous Testing
5. Continuous Monitoring
6. Continuous Feedback

DevOps Workflow

1. Sequential Job Execution
2. Parallel Job Execution
3. Branch Level Filtering
4. Fan -in /out in Repo

DevOps Principles

1. End to End responsibility
2. Continuous Improvement
3. Automate Everything
4. Custom Centric Action
5. Monitor And Test Everything
6. Works As One Team

DevOps Tools

1.Git 2. Jenkins 3. Maven 4. Docker 5. Kubernetes 6. Chef 7. Ansible 8. Puppet 9. Sensu 10. Salt Stack 11. Bamboo 12. Jira 13. Selenium 14. Nagios 15. Splunk 16. App Dynamic

DevOps Pipeline CI/CD

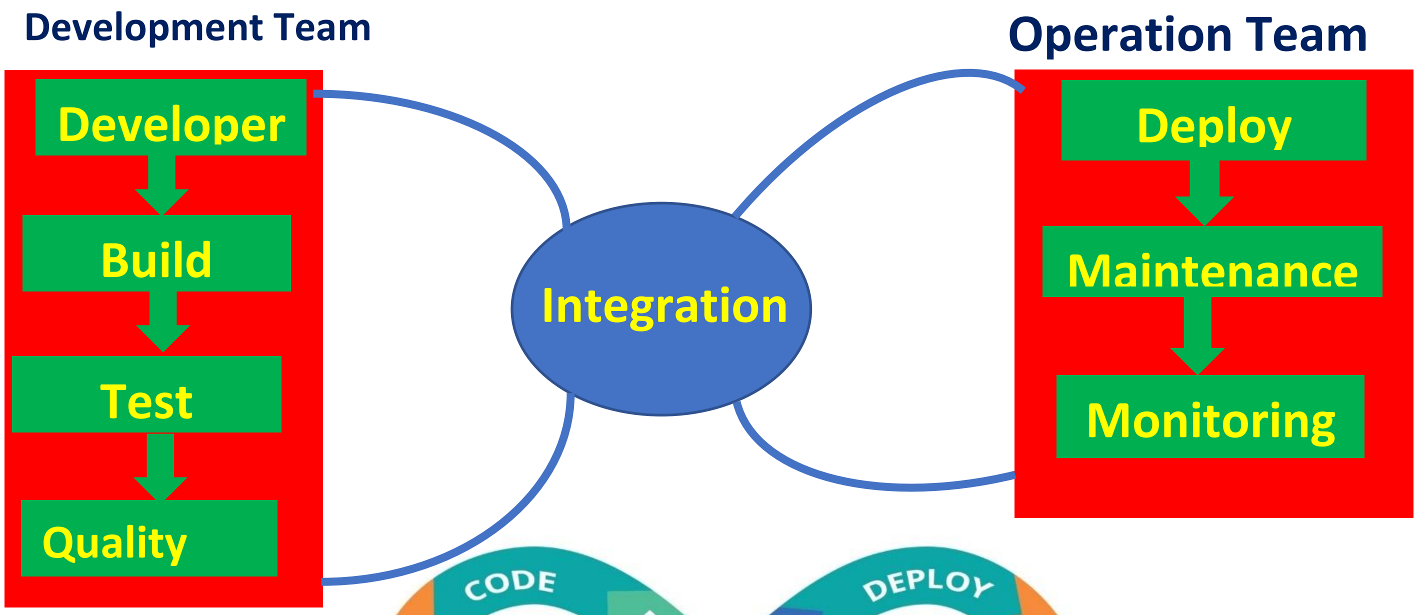
1. Source Control
2. Build Tools
3. Containerization
4. Configuration Management
5. Monitoring
6. Feedback

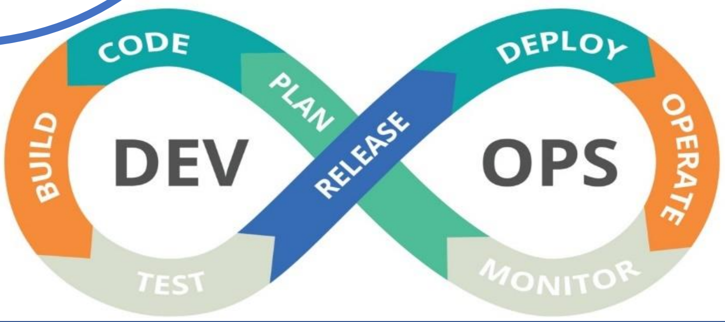
DevOps Methodology

1. Teams
2. Connectivity
3. Automation
4. On- Boarding Process
5. Project Environment
6. Shared Service
7. Naming Conventions
8. Defining Standards Role Across Team

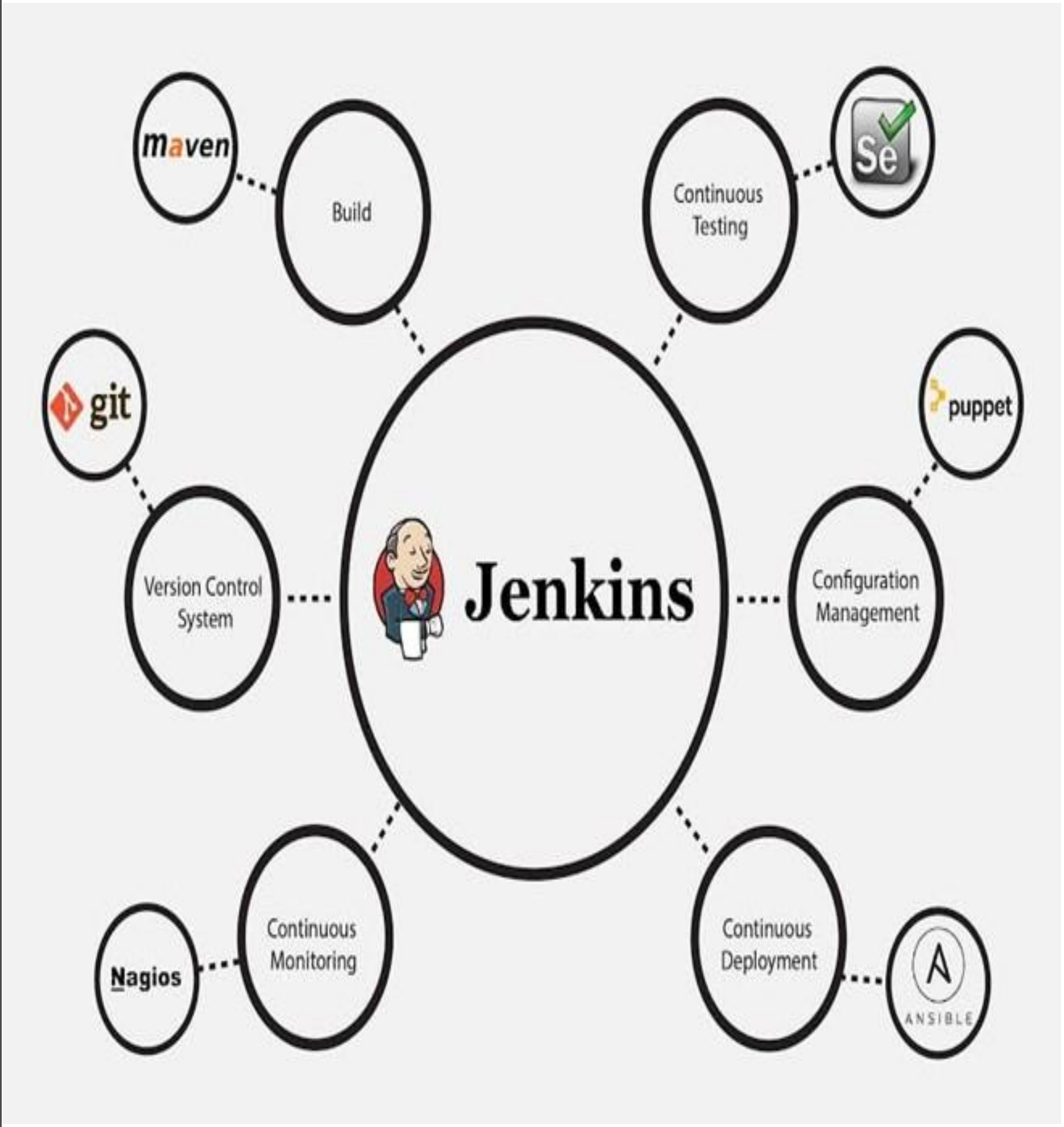
DevOps Automation Tools

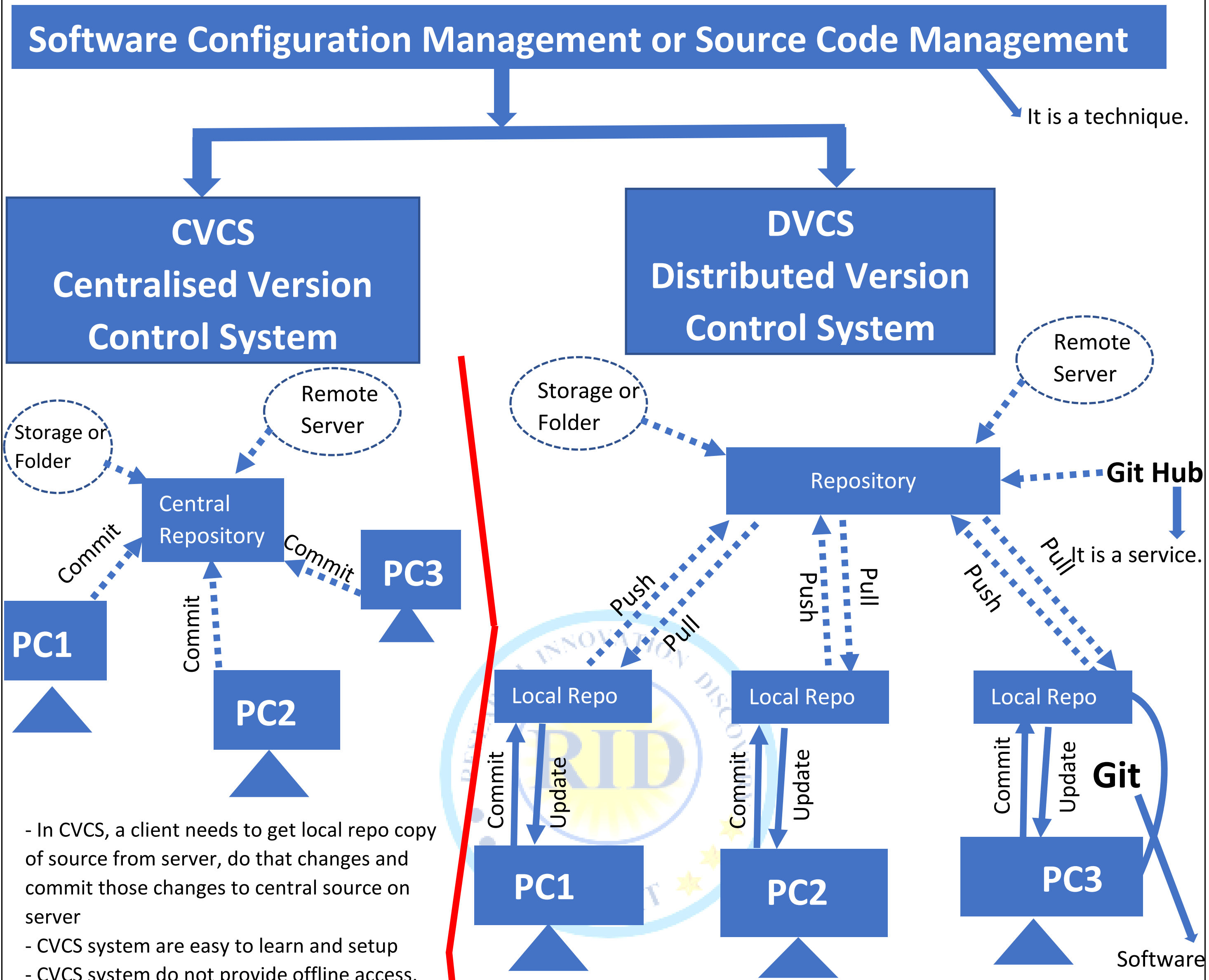
1. Infrastructure Automation (AWS)
2. Configuration Management (chef)
3. Deployment Automation (Jenkins)
4. Performance Management (App Dynamic)
5. Log Management (Splunk)
6. Monitoring (Nagios)





	Waterfall	Agile	DevOps
Basic philosophy	Systems are fully predictable and can be specified in advance. Assumes business needs remain broadly similar throughout project. Adjust schedule to preserve scope	Integrate business, dev and QA for rapid delivery of software. Iterative 'sprint' cycles. Assumes priority of business needs may change. Adjust scope to preserve schedule	Cross-functional teams utilize automation to enable continuous deployment of change. Constant feedback loop. Adjust scope to preserve schedule
Documentation level	Comprehensive	Light	Light
Automation level	Low	Varied	High
Delivery of value	Slow – only at major milestones (3-6 months)	Rapid (daily/weekly)	Continuous
Business ownership of project?	No (typical)	Yes	Yes
Response to new business needs (flexible requirements)	Extremely limited due to detailed specification	Responsive – iterative delivery enables prioritization	Highly responsive – cross-functional teams define business needs more precisely
Collaboration	Low – teams operate in functional silos	Improved – business is highly engaged, short dev cycles	High – all stakeholders involved from project start
Quality	Low – issues not identified until testing phase.	Improved – issues identified after every 'sprint'	High – automated unit testing during development
Risk	Increases as project progresses	Decreases as project progresses	Decreases as project progresses
Customer feedback	Infrequent - at project completion	Frequent – after every sprint	Continuous


maven
Build
Continuous Testing
Se
git
puppet
Version Control System
Jenkins
Configuration Management
Nagios
Continuous Monitoring
Continuous Deployment
ANSIBLE



- In CVCS, a client needs to get local repo copy of source from server, do that changes and commit those changes to central source on server
- CVCS system are easy to learn and setup
- CVCS system do not provide offline access.
- CVCS is slower as every command need to communicate with server.
- if CVCS server is Down developers cannot

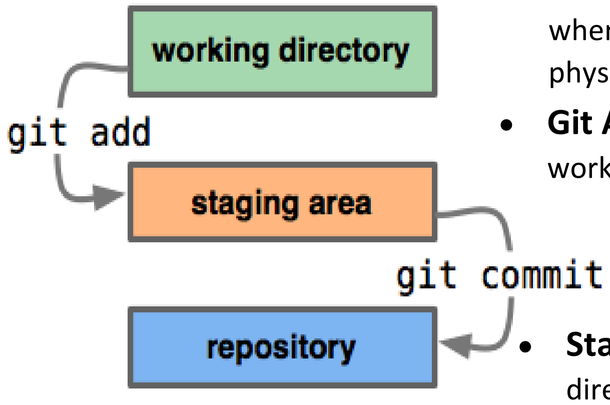
- In DVCS, each client can have a local repo as well and have a complete history on its client need to push the changes to branch which will then be pushed to source repository.
- Working on branches is easier in DVCS Developers faces less conflict.
- DVCS system are working fine on offline mode as a client copies the entire repository on their local machine.
- DVCS is faster as Mostly user deals with local copy without hitting server every time
- if DVCS server is Down developer can work using local Copies.

LVCS ---------> CVCS----------->DVCS

-Git is a distributes version control System Software Developed by Linus Torvalds in 7 Aprile 2005.

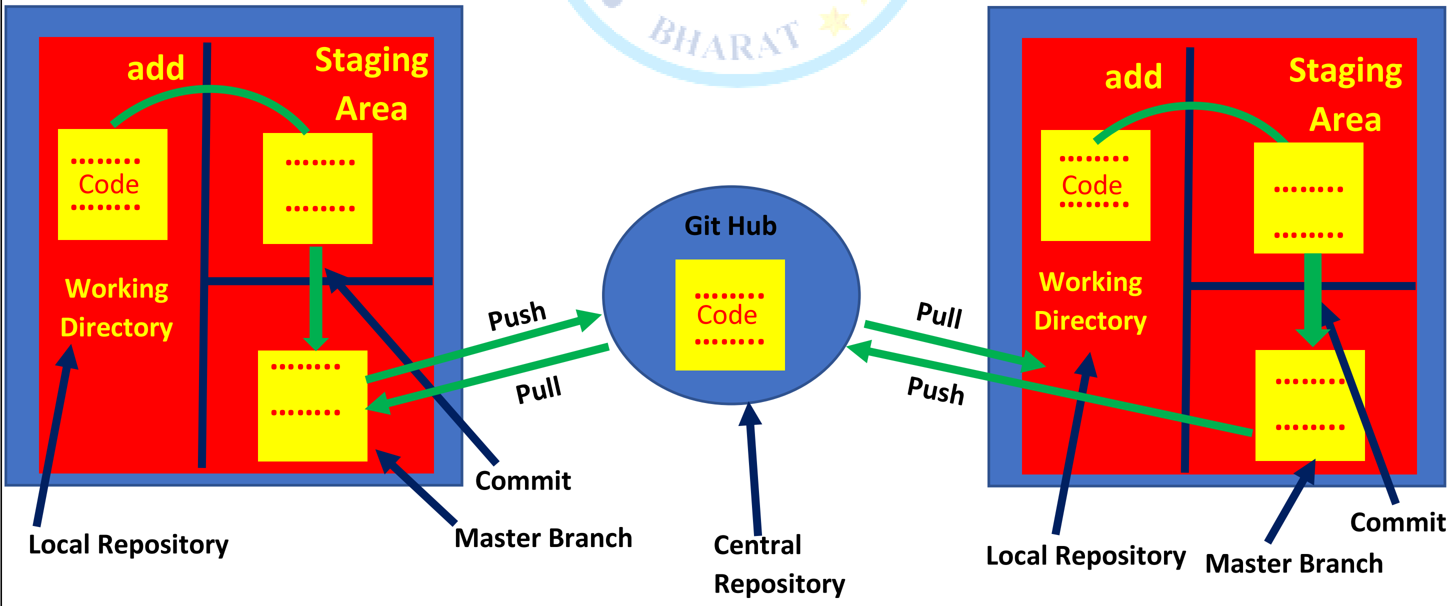
Linus Torvalds

➢ Stage of git

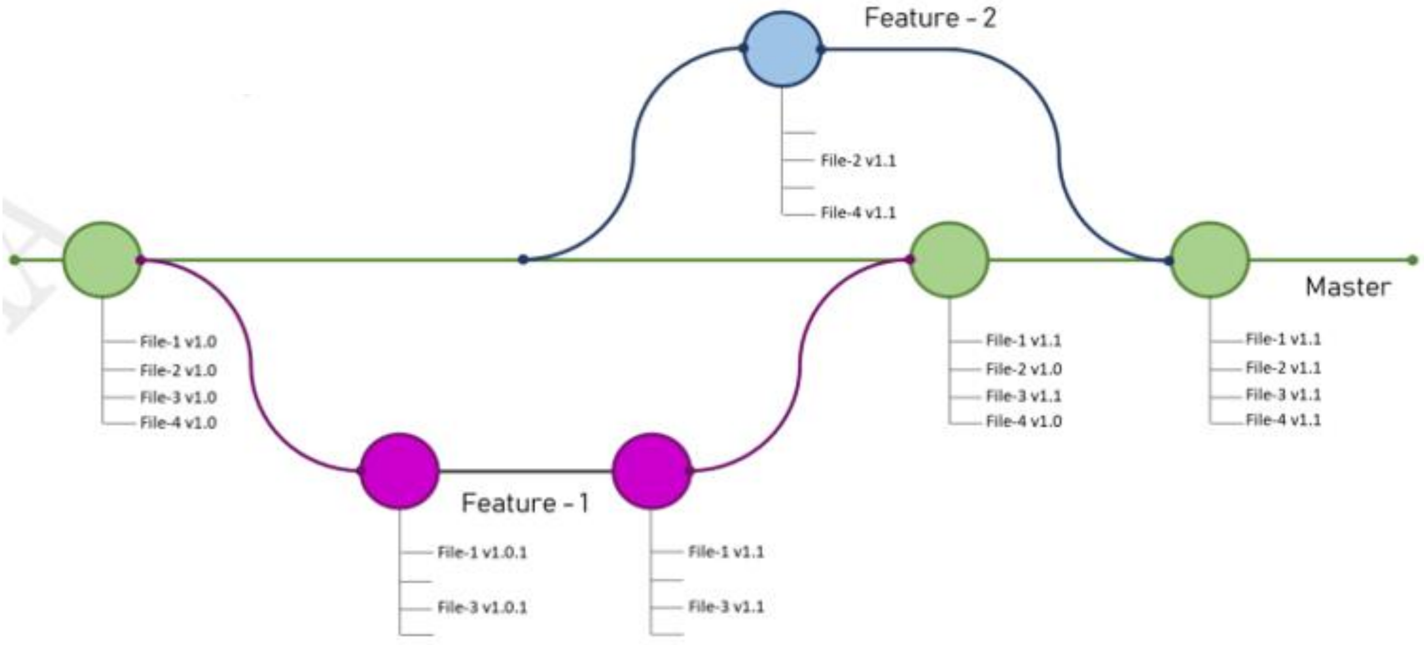


- **Working Directory: -**A working directory is a folder where you create to store all project's files. files see physically and can-do modification.
- **Git Add: -**The git add command adds a change in the working directory to the staging area.
- **Staging Area: -** The staging area is a file, contained Git directory, that stores information about what will go into next commit.
- **Git Commit: -** A commit is a snapshot of your entire repository at a specific time. Store changes in repository you will get one commit-id it is 40 alpha numeric character it uses SHA-1 checksum concept even if you change one dot commit-id will get change it actually help you to track changes commit is also named as SH1 hash.
- **Repository: -** Repository is a place where you have all your codes or kind of folder on server.

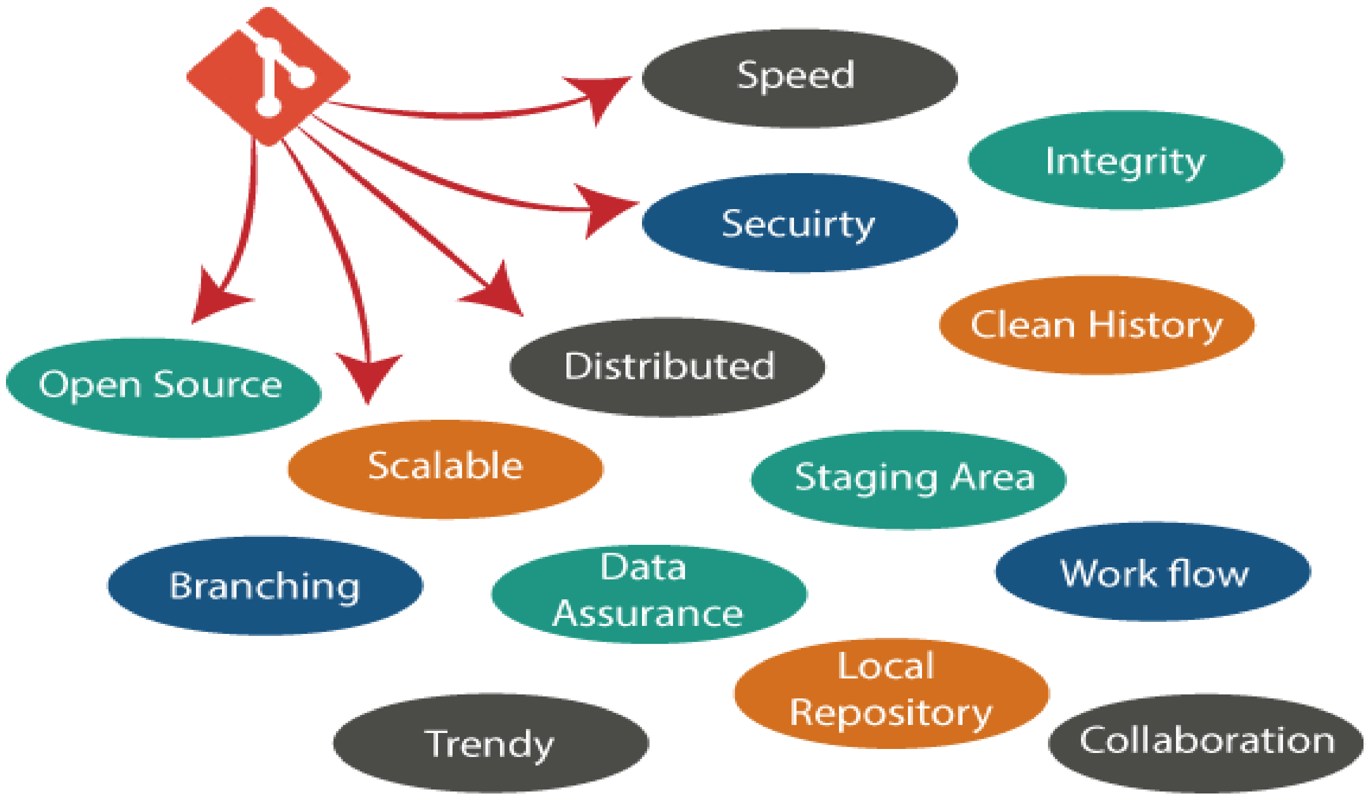
➢ Git Work Flow



1. **Repository: -**Repository is a place where you have all your codes or kind of folder on server. It is a kind of folder related to one project.
2. **Server: -**it stores all Repositories it's contains metadata also.
3. **Commit-ID/Version-ID/Version: -**Reference to identify each change to identify who changed.
4. **Tags: -**Tags assign a meaningful name with a specific version in the repository once a tag is created for a particular save even if you create a new commit, it will not be updated.
5. **Snapshots: -**Represents some data of particular time it is always incremental i.e it stores the changes (appended data) only not entire copy.
6. **Push: -**Push operations copies change from a local Repository instance to a Remote or central Repo. This is used to store the changes permanently into the git Repository.
7. **Pull: -**Pull operation copies the change from a Remote Repo to a local machine pull operation it is used for synchronisation between two Repo.
8. **Branch: -**Each task has one separate branch finally merges (code) all branches useful when you want to work parallelly. Can create one branch on the basis of another branch changes are personal to that particular branch
   - Default Branch is master
   - File created in workspace will be visible in any of the branch works space until you commit once you commit then that file belongs to that particular branch.



❖ Advantage of Git



❖ Types of Repositories

1. **Bare Repositories (Central Repo): -** Store and share only all central Repositories are bare Repo we cannot modify.
2. **Non-Bare Repositories (Local Repo): -** Where you can modify the files all local Repositories are non-bare repositories we can modify.

1.Basic Configuration

- git config --global user.name "Your Name": Sets the name for your commits.
- git config --global user.email "your.email@example.com": Sets email for your commits.
- git config --global core.editor "your_editor": Sets the default editor for Git.

Example:

1. Set the Name for Your Commits

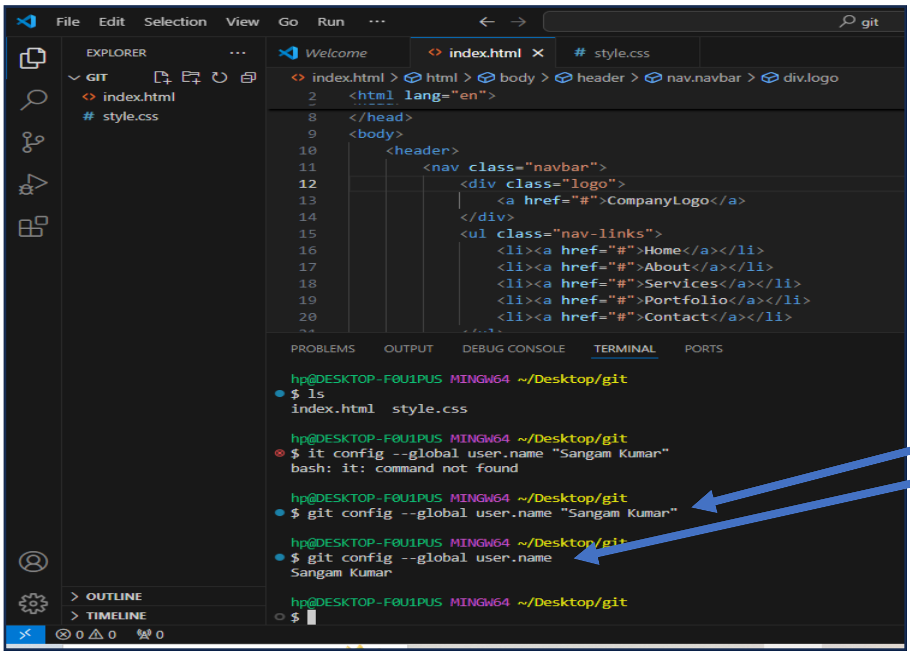
>> **git config --global user.name "Sangam Kumar"**

- This sets your name as "Sangam Kumar" for all commits across repositories.

2. How to check Name user

Example: hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/CSS (master)

- git config --global user.name
- Sangam Kumar


File Edit Selection View Go Run ···
git
EXPLORER
GIT
index.html
style.css
Welcome
index.html
style.css
index.html > html > body > header > nav.navbar > div.logo
OUTLINE
TIMELINE
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
<html lang="en">
</head>
<body>
    <header>
        <nav class="navbar">
            <div class="logo">
                <a href="#">CompanyLogo</a>
            </div>
            <ul class="nav-links">
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Portfolio</a></li>
                <li><a href="#">Contact</a></li>
```

```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ ls
index.html  style.css

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ it config --global user.name "Sangam Kumar"
bash: it: command not found

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ git config --global user.name "Sangam Kumar"

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ git config --global user.name
Sangam Kumar

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ 
```

3. Set the Email for Your Commits

- **git config --global user.email "ridorg.in@gmail.com"**
  - ➢ This sets your email to " ridorg.in@gmail.com " for all commits across repositories.

4. To check if your email ID is set in Git, you can use this command:

- **git config --global user.email**
  - ➢ If an email ID is set, Git will display it. If not, the command will return nothing.

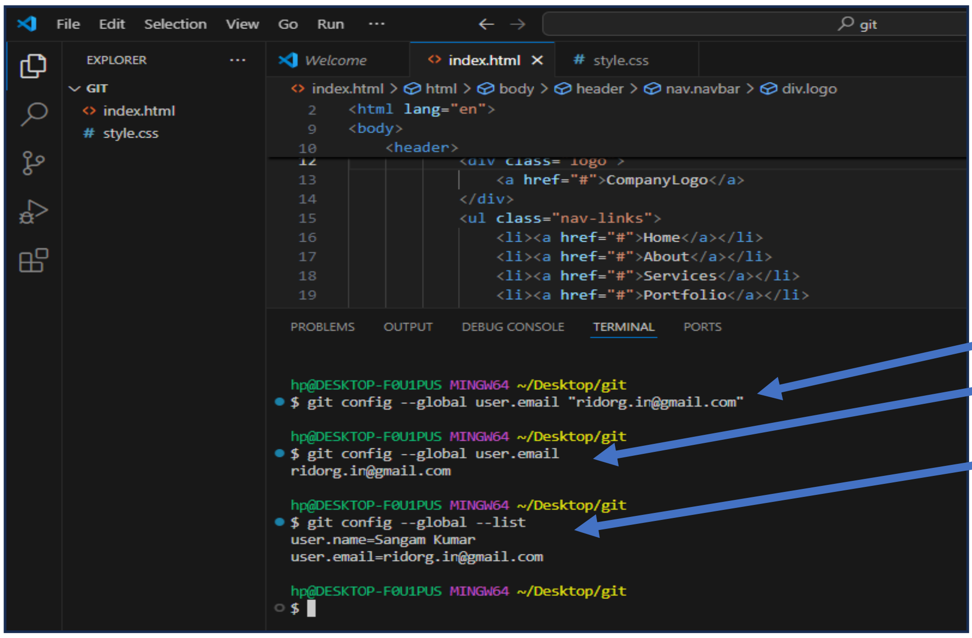
Note: Check All Global Configurations

  - ➢ Alternatively, to see all configurations, including user.email.

- **git config --global - -list**
  - ➢ This will list all global configurations, including both user.name and user.email.
  - ➢ hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/CSS (master)

```
$ git config --global --list
user.name=Sangam Kumar
```

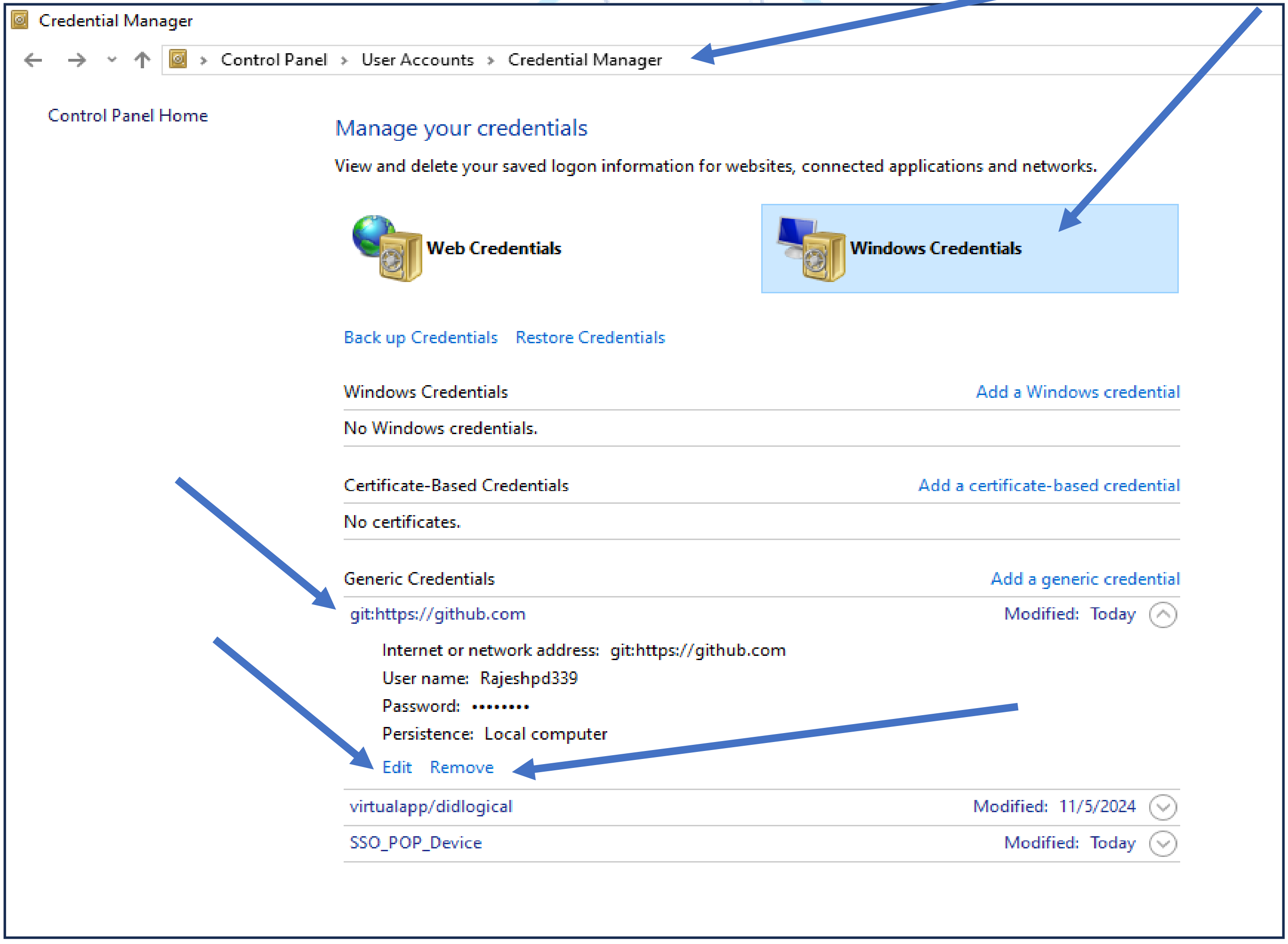
user.email=ridorg.in@gmail.com


File Edit Selection View Go Run ···
git
EXPLORER
GIT
index.html
style.css
Welcome
index.html
style.css
index.html > html > body > header > nav.navbar > div.logo
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
<html lang="en">
<body>
    <header>
            <div class="logo">
                <a href="#">CompanyLogo</a>
            </div>
            <ul class="nav-links">
                <li><a href="#">Home</a></li>
                <li><a href="#">About</a></li>
                <li><a href="#">Services</a></li>
                <li><a href="#">Portfolio</a></li>
```

```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ git config --global user.email "ridorg.in@gmail.com"

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ git config --global user.email
ridorg.in@gmail.com

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$ git config --global --list
user.name=Sangam Kumar
user.email=ridorg.in@gmail.com

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git
$
```

❖ Change user name and password in your local pc:



Credential Manager

Control Panel > User Accounts > Credential Manager

Control Panel Home

Manage your credentials

View and delete your saved logon information for websites, connected applications and networks.

Web Credentials | Windows Credentials

Back up Credentials Restore Credentials

Windows Credentials — Add a Windows credential

No Windows credentials.

Certificate-Based Credentials — Add a certificate-based credential

No certificates.

Generic Credentials — Add a generic credential

git:https://github.com — Modified: Today

Internet or network address: git:https://github.com

User name: Rajeshpd339

Password: ••••••••

Persistence: Local computer

Edit Remove

virtualapp/didlogical — Modified: 11/5/2024

SSO_POP_Device — Modified: Today

2. Set the Default Editor for Git

- By default, Git uses the system's default editor, which is usually Vim, but you can change this to a text editor you're more comfortable with, such as Visual Studio Code (VS Code).

❖ Why Set the Editor in Git?

- **Customizing the Workflow:** If you're not familiar with the default editor (Vim), changing it to an editor you're comfortable with can streamline your workflow.
- **Convenience & Consistency:** Setting up an editor like VS Code provides better syntax highlighting, auto-completion, & more user-friendly interface for writing commit msg.

❖ How to Set VS Code as the Default Editor for Git:

Step 1: Open the Command Line (Terminal)

- Open a terminal or command prompt on your system.

Step 2: Set VS Code as the Default Git Editor

- You can configure Git to use VS Code as the default editor by running this command:

➢ **git config --global core.editor "code --wait"**

- **git config --global:** This command sets the configuration globally for all your Git repositories (you can remove --global to set it for just the current repository).
- **core.editor:** This is the Git configuration option for setting the default editor.
- **"code --wait":** This is the command that tells Git to use Visual Studio Code. The --wait flag ensures that Git waits for you to close the editor before proceeding with the operation (like committing or rebasing).

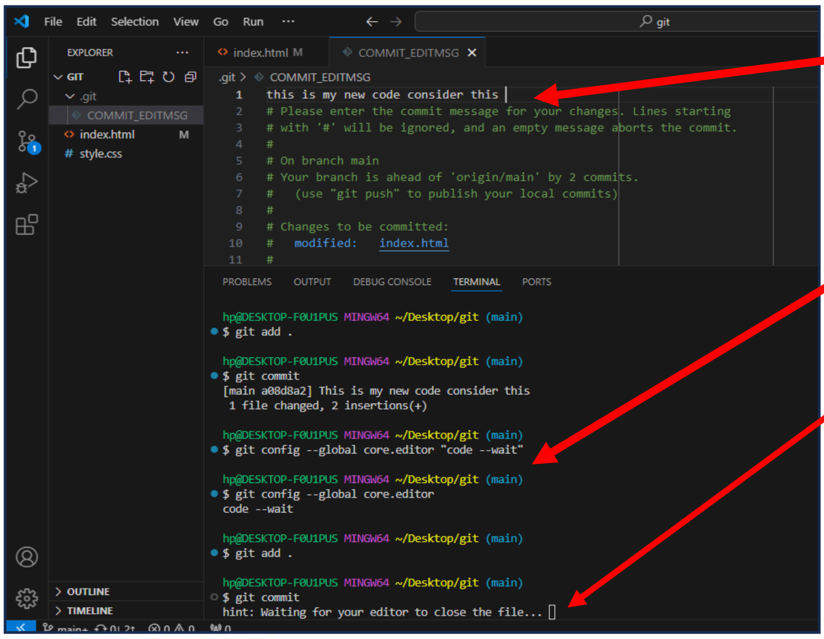
❖ Check VS Code is set as a default editor for this use this Command

➢ hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)

➢ **git config --global core.editor**

Step 3: Test the Editor Setting

- Change code then To test if your configuration worked, try to commit with a message:
- git commit
- This should open Visual Studio Code in a new window where you can enter your commit message. Once you save and close the file, the commit will be processed.



```
this is my new code consider this
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch main
# Your branch is ahead of 'origin/main' by 2 commits.
#   (use "git push" to publish your local commits)
#
# Changes to be committed:
#   modified:   index.html
#
```

```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git add .

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git commit
[main a08d8a2] This is my new code consider this
 1 file changed, 2 insertions(+)

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor "code --wait"

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor
code --wait

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git add .

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git commit
hint: Waiting for your editor to close the file...
```

This should open Visual Studio Code in a new window where you can enter your commit message. Once you save and close the file, the commit will be processed.

❖ Other Editors You Can Set

➢ If you'd like to set a different editor, you can use the following commands (replace "editor-command" with the respective editor's command):

Sublime Text:

➢ git config --global core.editor "subl -n -w"

Atom:

➢ git config --global core.editor "atom --wait"

Notepad++ (on Windows):

➢ git config --global core.editor "notepad++ -multiInst -nosession"

❖ Changing the Editor for a Specific Repository:

➢ If you only want to change the editor for a specific Git repository, omit the --global flag and run the command inside the repository directory:

➢ **git config core.editor "code --wait"**

Note: This will set VS Code as the default editor for that repository only.

❖ remove the default editor:

➢ To remove the default editor setting in Git, you can unset the core.editor configuration. Here's how to do it:

❖ Remove the Default Editor Setting Globally

➢ Run the following command to remove the global default editor configuration:

➢ **git config --global --unset core.editor**

➢ This will remove the editor setting from your global Git configuration, and Git will revert to using its default editor (usually Vim) or whatever is set in your system environment.

❖ Remove the Default Editor Setting for a Specific Repository

➢ If you only set the editor for a specific repository and want to remove it there (not globally), navigate to that repository in your terminal and run:

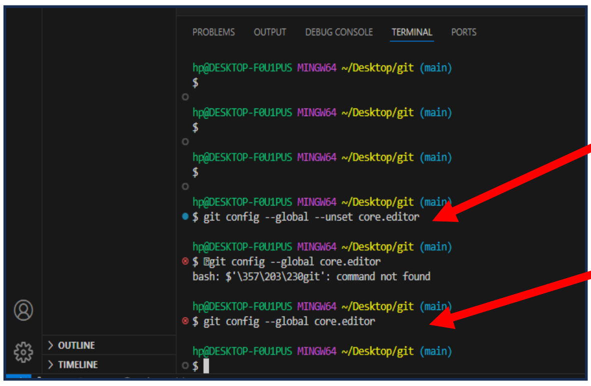
➢ **git config --unset core.editor**

❖ **Confirm the Editor Setting is Removed**

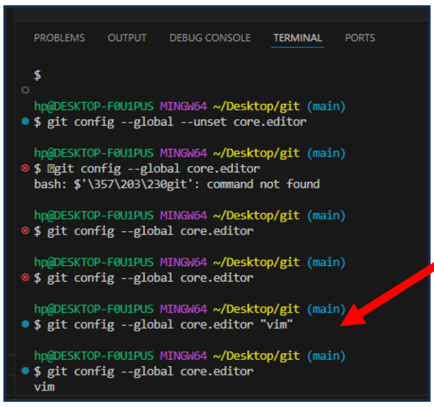
➢ To verify that the editor setting has been removed, you can check the configuration again:

➢ **git config --global core.editor**

➢ If the setting was successfully removed, this command should produce no output.



```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global --unset core.editor

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor
bash: $'\357\203\230git': command not found

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$
```



```
$

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global --unset core.editor

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor
bash: $'\357\203\230git': command not found

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor "vim"

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git (main)
$ git config --global core.editor
vim
```

Set Vim as the Default Git Editor Globally

3.Starting a Repository

- To start new Git repository, follow these steps and commands.

1. Initialize a New Git Repository

➢ To start a new repository in an existing project folder, navigate to the project directory in the terminal and use the following command:

➢ **git init**

➢ This creates a new .git folder in directory, which Git uses to track your files and history.

2. Add Files to the Staging Area

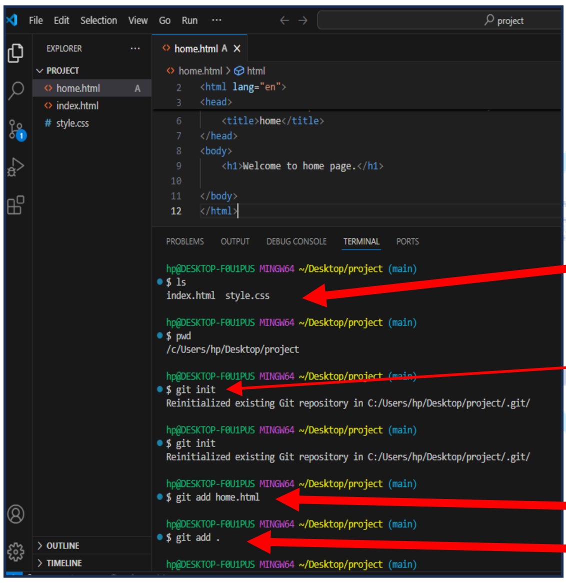
➢ After initializing the repository, you can add files to the staging area. The staging area is where you place files you want to include in the next commit.

❖ **To add a specific file:**

➢ **git add <file-name>**

❖ To add all files in the current directory:

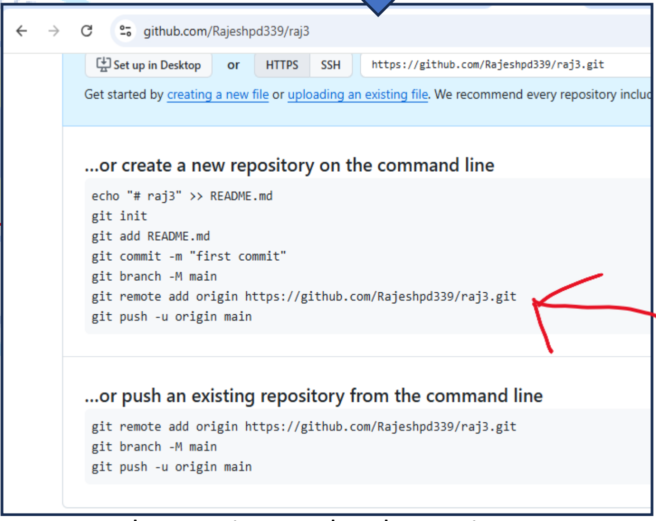
➢ **git add .**



```
<html lang="en">
<head>
    <title>home</title>
</head>
<body>
    <h1>Welcome to home page.</h1>

</body>
</html>
```

```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ ls
index.html  style.css

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ pwd
/c/Users/hp/Desktop/project

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git init
Reinitialized existing Git repository in C:/Users/hp/Desktop/project/.git/

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git init
Reinitialized existing Git repository in C:/Users/hp/Desktop/project/.git/

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git add home.html

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git add .

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
```

Create one repositor in your GitHub account
Example: repository name raj3
Then open raj3 repository then it will look like this


github.com/Rajeshpd339/raj3
Set up in Desktop
or
HTTPS
SSH
https://github.com/Rajeshpd339/raj3.git

Get started by creating a new file or uploading an existing file. We recommend every repository incluc

...or create a new repository on the command line

```
echo "# raj3" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/Rajeshpd339/raj3.git
git push -u origin main
```

...or push an existing repository from the command line

```
git remote add origin https://github.com/Rajeshpd339/raj3.git
git branch -M main
git push -u origin main
```

3. Make the First Commit

➢ Once files are staged, you can commit them to save changes in your local repository.

➢ **git commit -m "Initial commit"**

➢ The -m flag allows you to write a message describing the commit. It's a good practice to make your initial commit message "Initial commit".

4. Connect to a Remote Repository (Optional)

➢ If you want to link this local repository to a remote repository (like GitHub, GitLab, or Bitbucket), you'll need the remote URL.

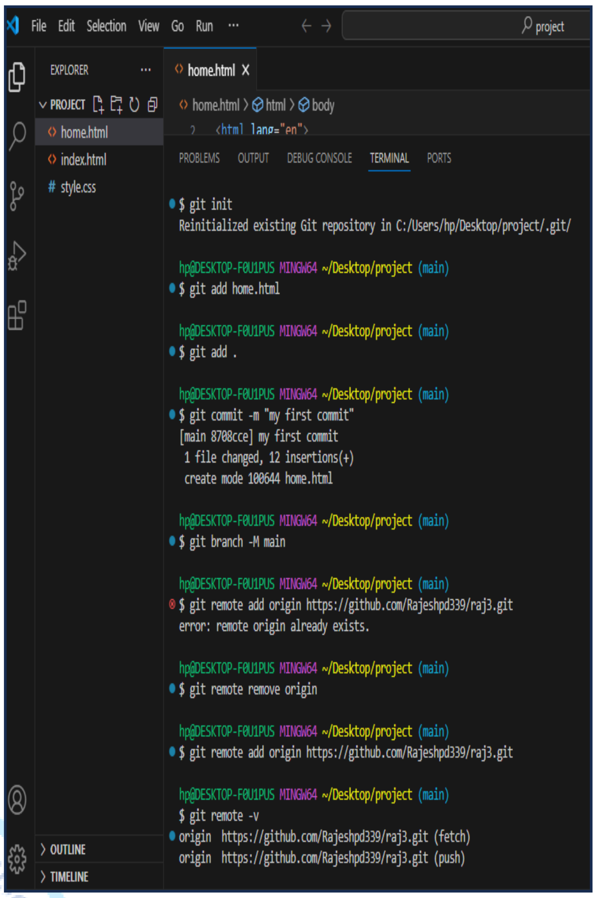
➢ **git remote add origin <repository-url>**

➢ Replace <repository-url> with the URL of your remote repository.

❖ **To verify that the remote was added successfully, use:**

➢ **git remote -v**

```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ ls
index.html  style.css
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ pwd
/c/Users/hp/Desktop/project
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git init
Reinitialized existing Git repository in C:/Users/hp/Desktop/project/.git/
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git init
Reinitialized existing Git repository in C:/Users/hp/Desktop/project/.git/
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git add home.html
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git add .
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git commit -m "my first commit"
[main 8708cce] my first commit
 1 file changed, 12 insertions(+)
 create mode 100644 home.html
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git branch -M main
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git
error: remote origin already exists.
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote remove origin
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote -v
origin  https://github.com/Rajeshpd339/raj3.git (fetch)
origin  https://github.com/Rajeshpd339/raj3.git (push)
```



```
  <html lang="en">
```

```
$ git init
Reinitialized existing Git repository in C:/Users/hp/Desktop/project/.git/

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git add home.html

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git add .

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git commit -m "my first commit"
[main 8708cce] my first commit
 1 file changed, 12 insertions(+)
 create mode 100644 home.html

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git branch -M main

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git
error: remote origin already exists.

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote remove origin

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote -v
origin  https://github.com/Rajeshpd339/raj3.git (fetch)
origin  https://github.com/Rajeshpd339/raj3.git (push)
```

❖ error message **"remote origin already exists"** indicates that a remote repository with name origin has already been set up for this Git repository. To resolve this, you have a few options:

Option 1: Remove the Existing Remote and Add a New One

If you want to remove the existing remote repository and then add a new one, use the following commands:

1. **Remove the Existing Remote**:
   - ➢ git remote remove origin
2. **Add the New Remote**:
   - ➢ git remote add origin https://github.com/Rajeshpd339/raj3.git

Option 2: Change the Existing Remote URL

If you just want to update the URL of the existing origin remote, you can use:

- ➢ git remote set-url origin https://github.com/Rajeshpd339/raj3.git

This changes the URL for origin without needing to remove it.

Option 3: Add a New Remote with a Different Name

If you want to keep the existing remote and add an additional one, give it a different name (e.g., origin2):

- ➢ git remote add origin2 https://github.com/Rajeshpd339/raj3.git

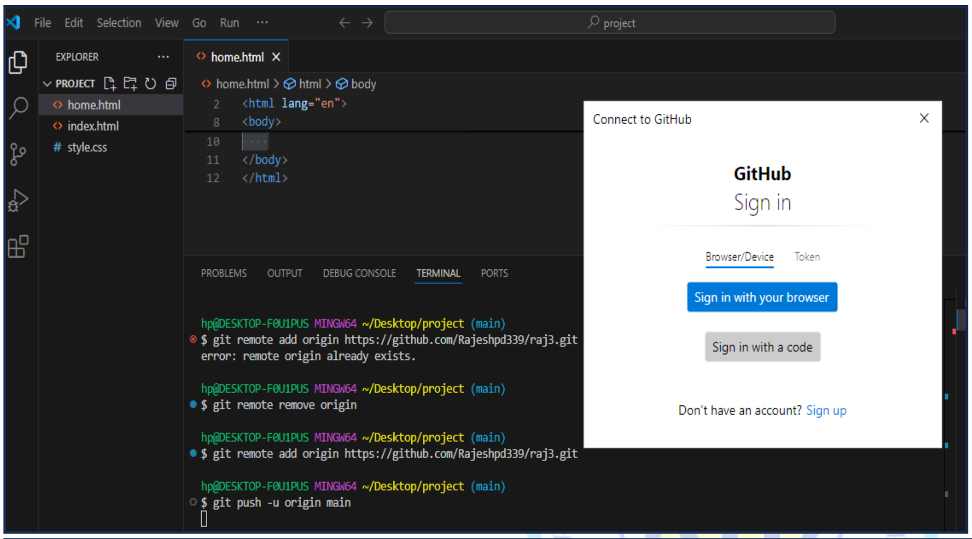
This way, you can use both remotes (origin and origin2) as needed.

5. Push the Local Repository to the Remote Repository (Optional)

➢ Once you've linked a remote, you can push your local commits to the remote repository.

➢ git push -u origin main

- The -u flag sets the origin as the default upstream for the main branch, so future pushes can be done with just git push.
- Replace main with master if your repository uses master as the default branch name.
- git init: Initializes a new Git repository in the current directory.
- git clone <repository_url>: Clones an existing repository from a URL.



```
<html lang="en">
<body>
...
</body>
</html>
```

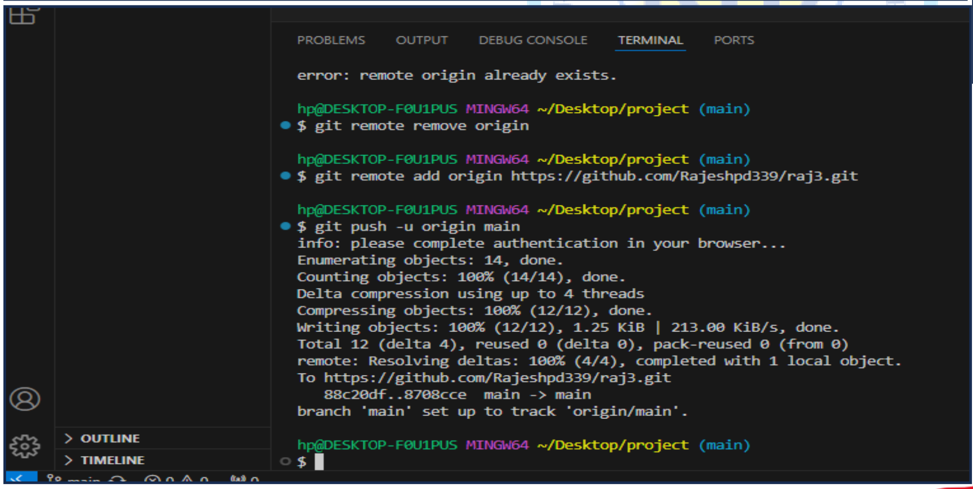
```
hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git
error: remote origin already exists.

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote remove origin

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git push -u origin main
```

How to Change the master Branch to main:

1. Rename the Local Branch:

If you're on the master branch, you can rename it to main using the following Git command:

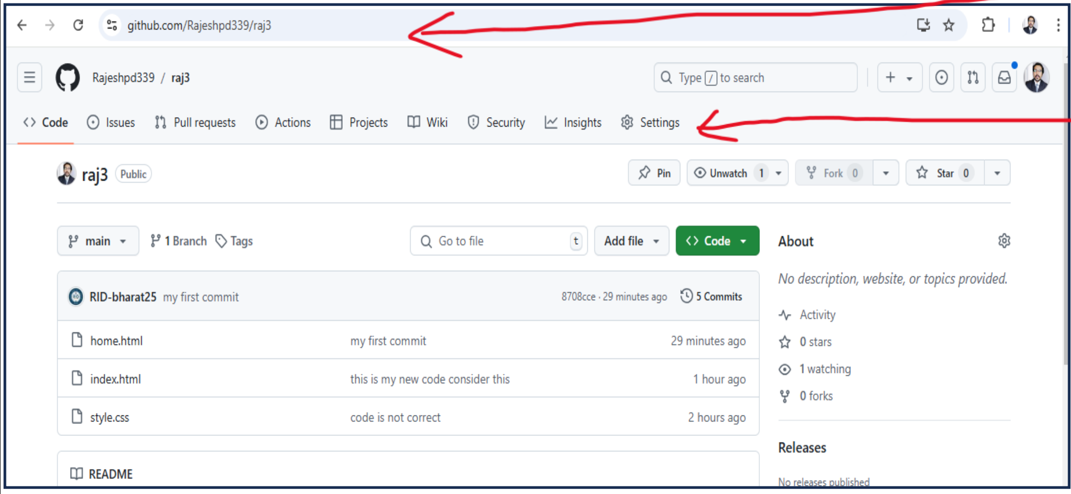
>> **git branch -m master main**



```
error: remote origin already exists.

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote remove origin

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git remote add origin https://github.com/Rajeshpd339/raj3.git

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$ git push -u origin main
info: please complete authentication in your browser...
Enumerating objects: 14, done.
Counting objects: 100% (14/14), done.
Delta compression using up to 4 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 1.25 KiB | 213.00 KiB/s, done.
Total 12 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 1 local object.
To https://github.com/Rajeshpd339/raj3.git
   88c20df..8708cce  main -> main
branch 'main' set up to track 'origin/main'.

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/project (main)
$
```

Open your GitHub and open your Raj3 Repo.

Open Setting for Deploy



RID-bharat25 my first commit — 8708cce · 29 minutes ago — 5 Commits

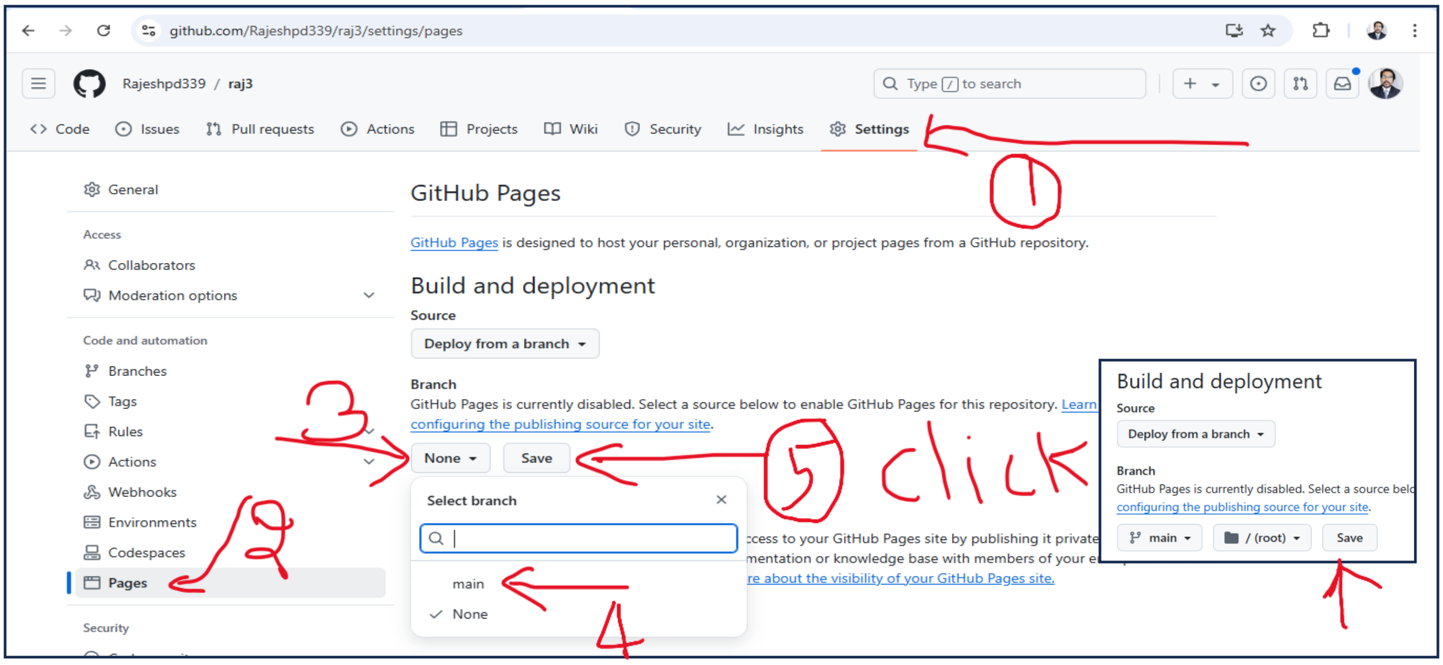
home.html	my first commit	29 minutes ago
index.html	this is my new code consider this	1 hour ago
style.css	code is not correct	2 hours ago

No description, website, or topics provided.

Activity
0 stars
1 watching
0 forks

Releases

No releases published



github.com/Rajeshpd339/raj3/settings/pages

Rajeshpd339 / raj3

Code · Issues · Pull requests · Actions · Projects · Wiki · Security · Insights · Settings

General

Access: Collaborators, Moderation options

Code and automation: Branches, Tags, Rules, Actions, Webhooks, Environments, Codespaces, Pages

Security

GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Build and deployment

Source

Deploy from a branch

Branch

GitHub Pages is currently disabled. Select a source below to enable GitHub Pages for this repository. Learn configuring the publishing source for your site.

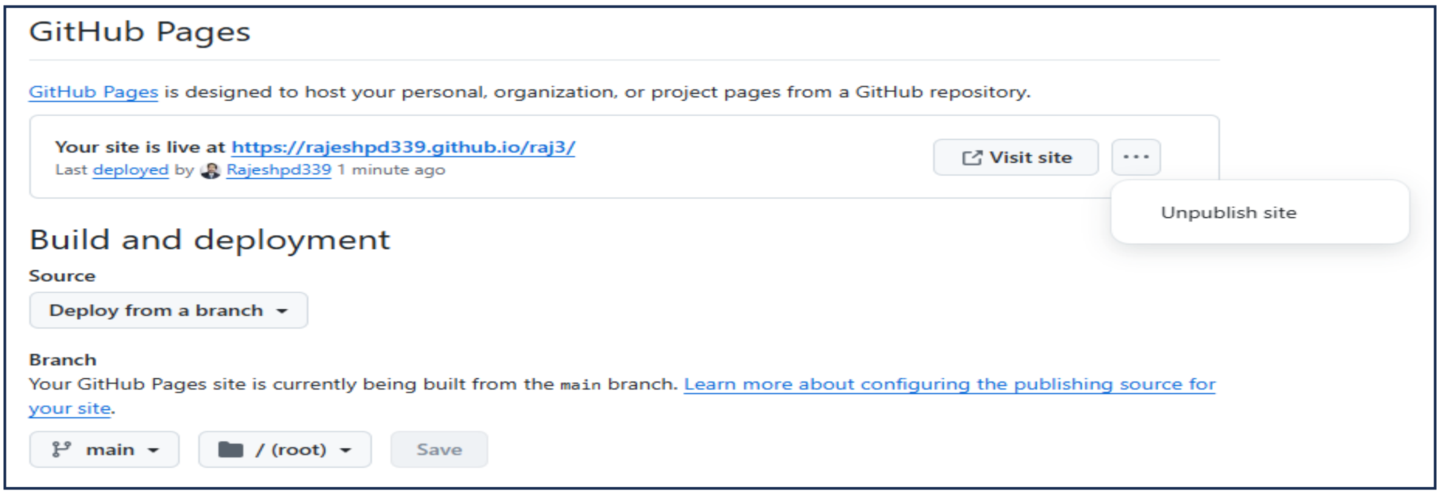
None · Save

Select branch: main, ✓ None

Build and deployment

Source: Deploy from a branch

Branch: main · / (root) · Save



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages from a GitHub repository.

Your site is live at https://rajeshpd339.github.io/raj3/

Last deployed by Rajeshpd339 1 minute ago

Visit site · ··· · Unpublish site

Build and deployment

Source

Deploy from a branch

Branch

Your GitHub Pages site is currently being built from the `main` branch. Learn more about configuring the publishing source for your site.

main · / (root) · Save

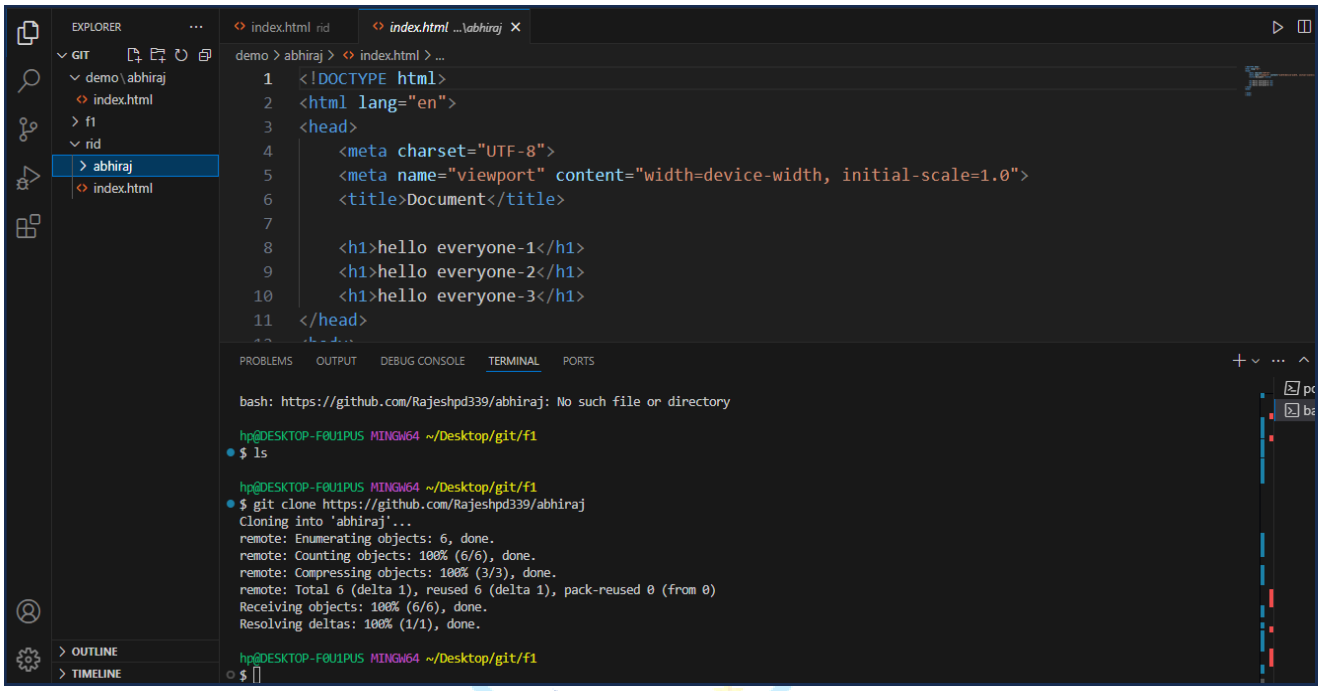
6. Clone the Repository (First-Time Setup)

- If this is the first time pulling the repository to your local machine:
  - **git clone <repository-url>**
- Replace <repository-url> with the HTTPS or SSH URL of the GitHub repository.

Example:

  - **git clone https://github.com/username/repo-name.git**
- This will download the repository into a directory named after the repository.

Example:



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>

    <h1>hello everyone-1</h1>
    <h1>hello everyone-2</h1>
    <h1>hello everyone-3</h1>
</head>
```

```
bash: https://github.com/Rajeshpd339/abhiraj: No such file or directory

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git/f1
$ ls

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git/f1
$ git clone https://github.com/Rajeshpd339/abhiraj
Cloning into 'abhiraj'...
remote: Enumerating objects: 6, done.
remote: Counting objects: 100% (6/6), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 6 (delta 1), reused 6 (delta 1), pack-reused 0 (from 0)
Receiving objects: 100% (6/6), done.
Resolving deltas: 100% (1/1), done.

hp@DESKTOP-F0U1PUS MINGW64 ~/Desktop/git/f1
$ 
```

git status

- git status command is used to display the state of the working directory and staging area in Git.

❖ git status Overview:

- When you run git status, Git provides information about the current branch, the changes that have been staged for commit, and the changes that have not been staged. It also shows if there are any untracked files that are not yet under version control.

❖ States in Git: there are four states in git

1. **Untracked files:** Files not yet added to Git (git add required).
2. **Unmodified files**: Files that haven't changed since the last commit (working tree is clean).
3. **Modified files:** Files that have changes but haven't been staged (git add required).
4. **Staged files:** Files that have been modified and staged for the next commit (git commit ready)

❖ Git tracks files in four different states:

1. Untracked:

- Files that have not been added to Git (i.e., they are new and not being tracked by Git).
- These files need to be explicitly added to the repository using git add.

Example: - You create a new file newfile.txt, but you haven't added it yet.

```
>> git status
```

Output:

```
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    newfile.txt
```

2. Unmodified:

- Files that have not been changed since the last commit. They are exactly the same as in the previous commit.
- These files do not need to be added to the staging area because no changes were made.

Example: After a commit, you check the status and find that files have not changed.

```
>> git status
```

Output:

```
On branch main
nothing to commit, working tree clean
```

3. Modified:

- Files that have been modified but not yet staged for commit. You need to stage them using git add before committing.
- These are files that you have edited but haven't added to Git's staging area.

Example: You modify file1.txt, but you haven't added it yet.

```
>> git status
```

Output:

```
On branch main
  Changes not staged for commit:
 (use "git add <file>..." to update what will be committed)
  modified:   file1.txt
```

4. Staged

- Files that have been modified and staged (added using git add). They are ready to be committed in the next commit.
- These files have been marked for inclusion in the next commit.

Example: You modify file1.txt and then run git add file1.txt.

```
>> git status
```

Output:

```
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   file1.txt
```

7.Creating the branch:

Summary of Commands	
Task	**Command**
Create a branch	git branch <branch-name>
Switch to a branch	git checkout <branch-name> / git switch
Create and switch to a branch	git checkout -b <branch-name>
List branches	git branch -a
Push a branch to GitHub	git push origin <branch-name>
Delete a local branch	git branch -d <branch-name>
Delete a remote branch	git push origin --delete <branch-name>

Step-by-Step Example:

1. Clone the Repository

- Start by cloning an existing repository or initializing a new one.
  - **git clone <repository-url>**
  - **cd <repository-name>**

Example:

- **git clone https://github.com/username/repository-name.git**
- **cd repository-name**

2. Create a New Branch

- Use the git branch command to create a new branch.
- **git branch <branch-name>**

Example: git branch feature-login

3. Switch to the New Branch

- Switch to the branch using git checkout or git switch.
- **git checkout <branch-name>**

  # OR
- **git switch <branch-name>**

Example: git checkout feature-login

4. Create and Switch in a Single Command

- You can create and switch to a branch in one step:
- **git checkout -b <branch-name>**

Example: git checkout -b feature-dashboard

5. Verify the Current Branch

- Check which branch you are currently on:
- **git branch**
- The current branch will be highlighted with an asterisk (*).

6. Push the New Branch to GitHub

- Push the branch to the remote repository on GitHub:
- **git push origin <branch-name>**

Example: git push origin feature-login

7. List All Branches

- To list all branches (local and remote):
  - **git branch** # Lists local branches
  - **git branch -r** # Lists remote branches
  - **git branch -a** # Lists all branches (local and remote)

8. Switch Between Branches

- Switch to another branch to continue working:
  - **git checkout <branch-name** # OR
  - **git switch <branch-name>**

Example:

- git checkout main

9. Delete a Branch

- After merging or when no longer needed, delete a branch:
- **Delete Local Branch**:
  - git branch -d <branch-name>

Example:

- git branch -d feature-login
- **Delete Remote Branch:**
- git push origin --delete <branch-name>

Example: git push origin --delete feature-login

❖ Practical GitHub Workflow

1. **Clone a Repository**:
   - git clone https://github.com/username/project-name.git
   - cd project-name
2. **Create and Work on Branches:**
   - **git checkout -b feature-1**
     # Make changes
   - git add .
   - **git commit -m "Add feature 1"**
   - git push origin feature-1
3. **Switch to Another Branch:**
   - **git checkout -b feature-2**

# Make changes
- git add .
- **git commit -m "Add feature 2"**
- git push origin feature-2

4. **Merge Changes in GitHub:**
    - Open a pull request for feature-1 and feature-2 to merge them into the main branch.
    - After merging, you can delete the branches.

❖ git merge raj

➢ Step-by-Step Guide

1. Make Sure You Have Latest Changes

- Before proceeding, ensure your raj branch is up to date and contains the changes you want to push.
    - Switch to raj branch (if you aren't already):
    - ➢ **git checkout raj**
    - Push any local changes in raj (if any):
    - ➢ **git push origin raj**

2. Switch to the main Branch

- Now, switch to the main branch where you want to merge the changes.
    - ➢ git checkout main

3. Pull the Latest main (Optional but Recommended)

- It's always a good idea to pull the latest changes from the main branch to avoid any
    - ➢ **git pull origin main**

4. Merge raj into main

- **Now that you're on the main branch, merge the raj branch into main:**
    - ➢ **git merge raj**
    - If there are no conflicts, this will complete the merge.
    - If there are conflicts, Git will prompt you to resolve them manually. After resolving the conflicts, add the resolved files and commit the merge.

5. Push the Changes to GitHub

- Finally, push the changes in the main branch (which now includes the changes from raj) to GitHub:
    - ➢ **git push origin main**

Example Walkthrough

1. **Switch to raj branch (if you're not on it already):**

    git checkout raj

2. **Push raj branch (if you haven't pushed it yet):**

    git push origin raj

3. **Switch to main branch:**

    git checkout main

4. **Pull the latest changes from main:**

    git pull origin main

5. **Merge raj into main:**

    git merge raj

6. **Push the merged main branch to GitHub.**

git push origin main

Summary of Commands

Task	**Command**
Switch to raj branch	git checkout raj
Push raj branch to GitHub	git push origin raj
Switch to main branch	git checkout main
Pull latest changes from main	git pull origin main
Merge raj into main	git merge raj
Push merged changes to GitHub	**git push origin main**

❖ How to ignore some files while committing

Create one hidden file with .gitignore extension and enter file format which you want to ignore

Example: - Vi file name like

```
 Vi anjraj.gitignore
 Enter .css
          .java
Esc  :wq
```

```
#git add anjraj.gitignore
# git commit -m  "write some text"
#git statu
```

Create some text.java & .cs files and add them by running "git add"

Example: -

```
touch file1.txt file2.txt file3.java file4.cs
#ls
# git status
# git add
#git status
#git commit -m "my first commit"
```

- **Each task has one separate branch after done with code merge other branches with master this concept is useful for parallel development, we can create any no of branches**
- When created new branch data of existing branch is copied to new branch
- Command
- First to see list of available branch git branch
- Create a new branch
- # git branch <branch name>
- # git branch branch1
- To see branch command
- # girt branch
- To switch Branch
- # git checkout <branch name>
- #git checkout branch1
- # git log - - online
- To delete branch
- # git branch - d brach1

❖ **Merge**

- You can't merge branch of different Repositories we use pulling mechanism to merge branches

Command

- # git merge <branch name>
- to verify the merge
- # git log
- To Push to central Repo like git hub
- # git push origin master

❖ **Git Conflict: -**

When same name file having different content in different branches if you do merge conflict occurs (resolve conflict) then add and commit

- Conflict occurs when you merge branches

❖ **Git stashing: -**

- Suppose you are implementing a new feature for your product your code is in progress and suddenly a customer escalation comes because of this you have to keep aside your new feature work for few hours you can not commit your partial code and also can not throw away your changes so you need some temporary storage when you can store on commit it
- To stash an item (only applies to modified files not new files)

Command

To stash an item

# git stash

- To see stashed items

# git stash apply stash {0}

- Then you can add and commit

# git stash clear.

❖ **Git Reset: -**

- Git reset is a powerful command that is used to undo local changes to the state of a git repo
- To reset staging area

# git reset <filename>
# git reset .

- To reset the changes from both staging are and working directory at a time

# git reset -- hard

❖ **Git revert: -**

- The revert command helps you undo an existing commit
- It does not delete any data in this process instead rather git create a new commit with the included files reverted to their previous state so you version control history moves forward while the state of your file moves backword
  Reset ---> before commit
  Revert-----> after commit
- To reset staging area

❖ **Command**

```
$sudo su
#cd directory name
#ls
# git status
# cat newfile ( create one new file and write code)
#git add .
# git commit -m "write any message"
#git log – oneline
#git revert <commit-id>
```

❖ How to Remove untracked files: -

```
# git clean -n (day run)
# git clean -f (forcefully)
```

❖ **Tags**

- Tag operation allows giving meaningful names to a specific version in the repository
- To apply tag
- #git tag -a <tag name > -m <message> <commit-id>
- To see the list of tags
- # git tag
- To see particular commit content by using tag
- #git tag show <tag name>
- To delete a tag
- #git tag -d <tagname>

Git important Command

1.How to set the user name

```
>> git config --global user.name "Your Name"
```

2.How to check the user name

```
>>   git config --global user.name
```

3.How to set the user Email

```
>> git config --global user.email "ridorg@gmail.com"
```

4.How to check the user Email

```
>> git config --global user.email
```

5.How to check the both usr name and use email

```
>> git config --global --list
```

6.How to delete the user’s name

```
>> git config --global --unset user.name
```

7.How to delete the user’s Email

```
>> git config --global --unset user.email
```

8.How to delete the both user.name and user. Email

```
>> git config --global --unset user.name && git config --global --unset user.email
```

9.Check the Configuration

```
>> git config --global –list
```

10. How to clear the Git Bash terminal

```
>> clear or reset
```

How Set the Default Editor for Git

1.How to check the default text editor set in Git

```
>> git config --global core.editor  or
```

2.How to Check Full Git Configuration

```
>> git config --global --list
```

3.How to set the vs code default text editor

```
>> git config --global core.editor "code --wait"
```

4.How to delete the vs code as default text Editor

```
>> git config --global --unset core.editor
```

5.How to Verify the Change

```
>> git config --global --list
```

6.How to Set a New Default Editor

```
>> git config --global core.editor "vim"
```

7.How to Set Nano:

```
>> git config --global core.editor "nano"
```

8.How to Set Notepad (Windows Only):

```
>> git config --global core.editor "notepad"
        git init
        ls -a
        git status
        git add .
        git commit -m "IniƟal commit"
        git remote add origin
        hp://github.com/username/repository.git
        git branch -m master main
        git branch -m main master
        git branch -M main
        git remote -v
        git remote remove origin
        git push -u origin main
```

Git Branch Command

1. How to create a New Branch?

- **Syntax:** git branch <branch-name>
- **Example:** git branch login

2. How to List All Branches?

Example: git branch

3. Hoe to Switch to Another Branch

syntax: git checkout <branch-name> or
Syntax: git switch <branch-name>
Example: git checkout login
Example: - git switch login

4. How to Create and Switch to a New Branch

- **Syntax:** git checkout -b <branch-name>
- **Syntax:** git switch -c <branch-name>
- **Example:** git checkout -b newbranch
- **Example:** git switch -c newb2

5.How to add some feactures in created data branch and push to in github

- **Syntax:** git push origin <branch_name>
- **Example:** git push origin f2

6. Show Differences Between Two Branches

- **Syntax:** git diff <branch-1> <branch-2>
- **Example:** git diff main f2
- **Example:** git diff f1

7.How to merge the code from one branch to another or main branch

- **Syntax:** git merge <source-branch>
- **Example:** git merge main
- **Example:** git merge f2 # Merge f2 into main

1. Switch to the main branch

- git checkout main

2. Merge the f1 branch into main

- git merge f1

5. How to delete the brahcn

- **syntax:** git brach -d <branch_name>
- **Example:** git branch -d f1

6. How to delete the branch without merge Force Delete the Branch

- **Syntax:** git branch -D <Branch_Name>
- **Example:** git branch -D f1

❖ What is a Pull Request (PR) in Git?

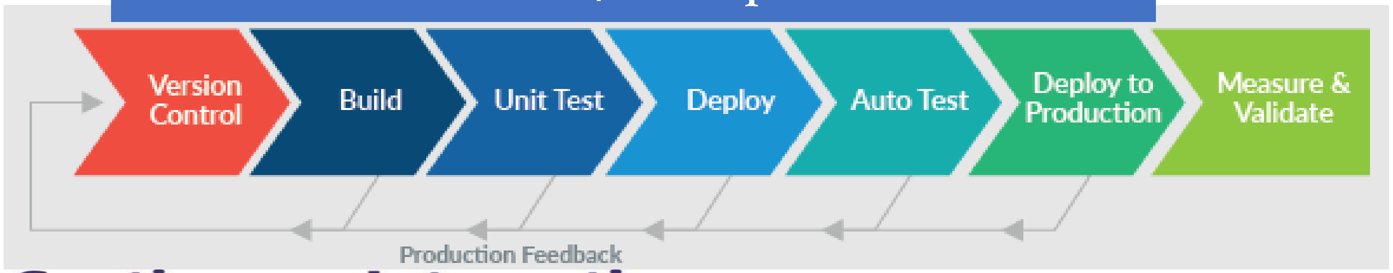
- Pull Request (PR) is a Git feature on platforms like GitHub, GitLab, and Bitbucket that lets developers propose, review, and merge changes from one branch to another before merging. It ensures collaboration, code review, and version control

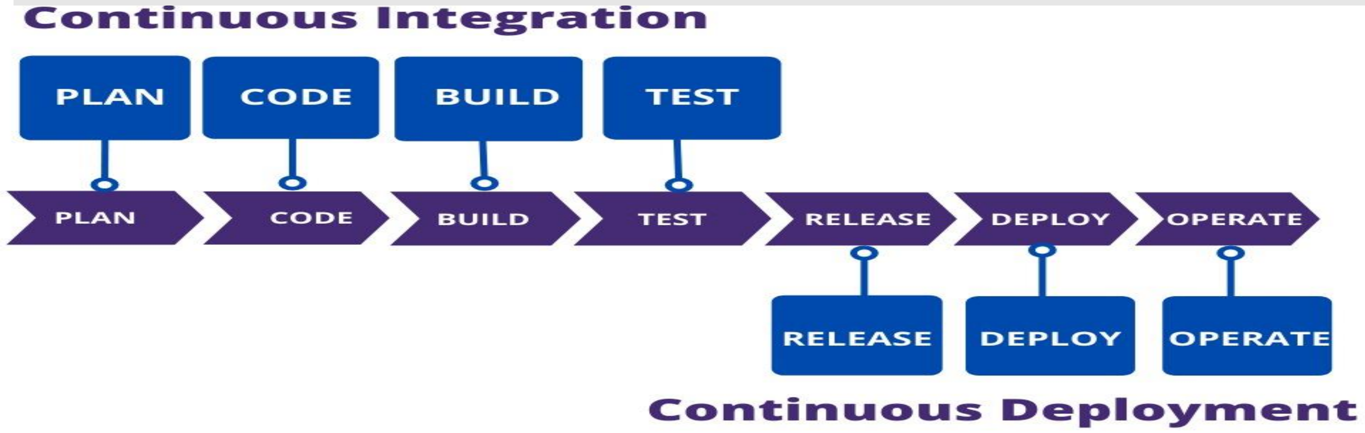
❖ What is a Pull Command?

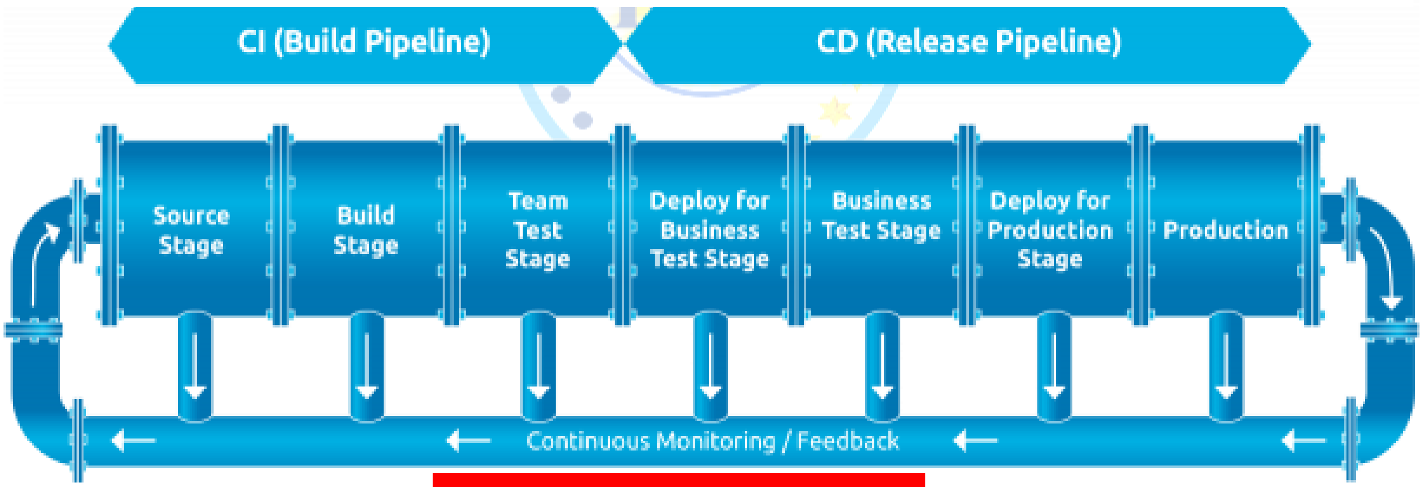
- git pull command fetches the latest changes from a remote repository and merges them into your local branch.
- **Syntax:** git pull <remote> <branch>
- **Example:** git pull origin main

Jenkins

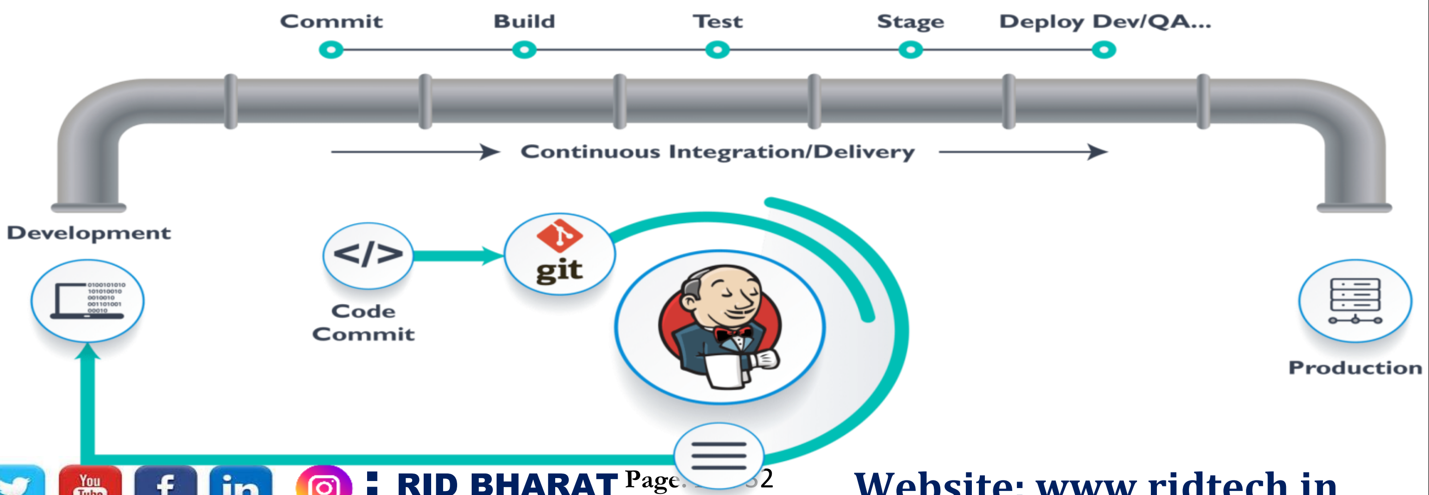
What is CI/CD Pipelines







Jenkins Work Flow

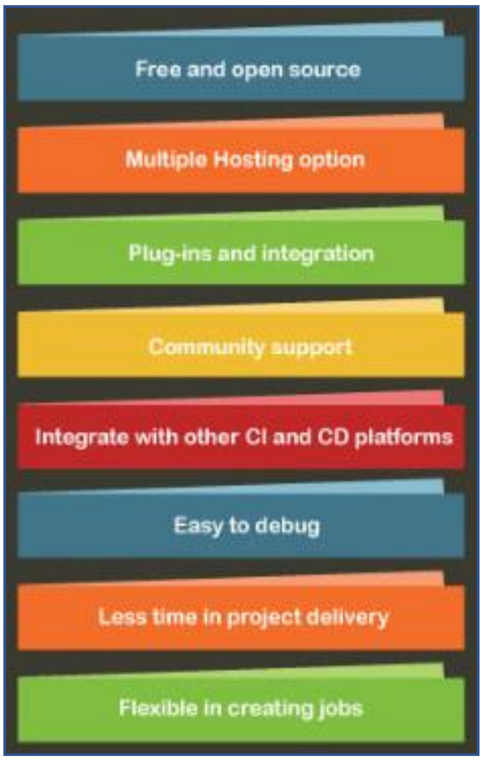


tholody

:en Java run on all OS, Port no=8080 Free Community Support and Might be 1ST Choice tool for CI ,
re SDLC, Developed by Sun Micro System in 2004

Maven, Selenium and Artifactory Plugins to Jenkins
:s code in git Hub, Jenkins Pull that code & send to maven for build
Jenkins pull that code and send to selenium for testing .
ne then Jenkins will pull that code and send to antifactory as archive purpose requirement and so on
y with Jenkins

Advantage of Jenkins



ree: it is free of cost. It is platform-independent.
ation: It has its type of plug-in, which helps the developer a lot in executing
lug-ins can be developed by anyone and for anyone.
n be installed on any operating system You can also run Jenkins on the cloud
id deploying it on a VM. You can also use a Docker container in it.
rt: Jenkins has great support from the developer community.
ther CI/CD platforms: Jenkins supports many CI/CD platforms, not only the
ke interaction with other tools also. Several plug-ins are available in it, which
ke connections with other CI/CD platforms.
sync: Jenkins focuses on a centralized way of working. All the members of the
.
very easy to find out the errors in the Jenkins. The developer can easily check
e it.
r the project: It happens because of its continuous integration feature.
the jobs: It is very flexible in creating the jobs. It can create jobs both in
pipeline process very easily.
gement (SCM): Jenkins supports different types of source code repositories

ss of converting in GUI from CLI very easy.
e data support to project management.
anguages, like Java, Python, etc.

Step to work Jenkins

1.Download Git

2. Download Java

3.Download Maven

4.Download Jenkins

5.Open Localhost:8080

Scheduled Project

- Click an any project -> Configure ->Build triggers ->Build periodically
- * * * * * (Mintus hr day month week) ->save
- Can see automatic builds after every 1 Mintus
- You can manually trigger build as well.

Source Code Polling (Poll SCM)

- Now go to Jenkins home page
- go to any project ->Configure
- Now go to Build trigger
- Poll scm
- Schedule * * * * * (Mintus hr day month week) ->save
- Now go to Git hub account -> do some changes in code->commit change
- You can see after 1 Mintus it build automatically

User Management

- Go to Jenkins homepage -> manage Jenkins ->manage users
- Create two users -> anjraj & raj
- Now login as -> anjraj
- {by default you have all the permission }
- Login as “admin” again
- Go to manage Jenkins->manage plugins
- Search “Role-based

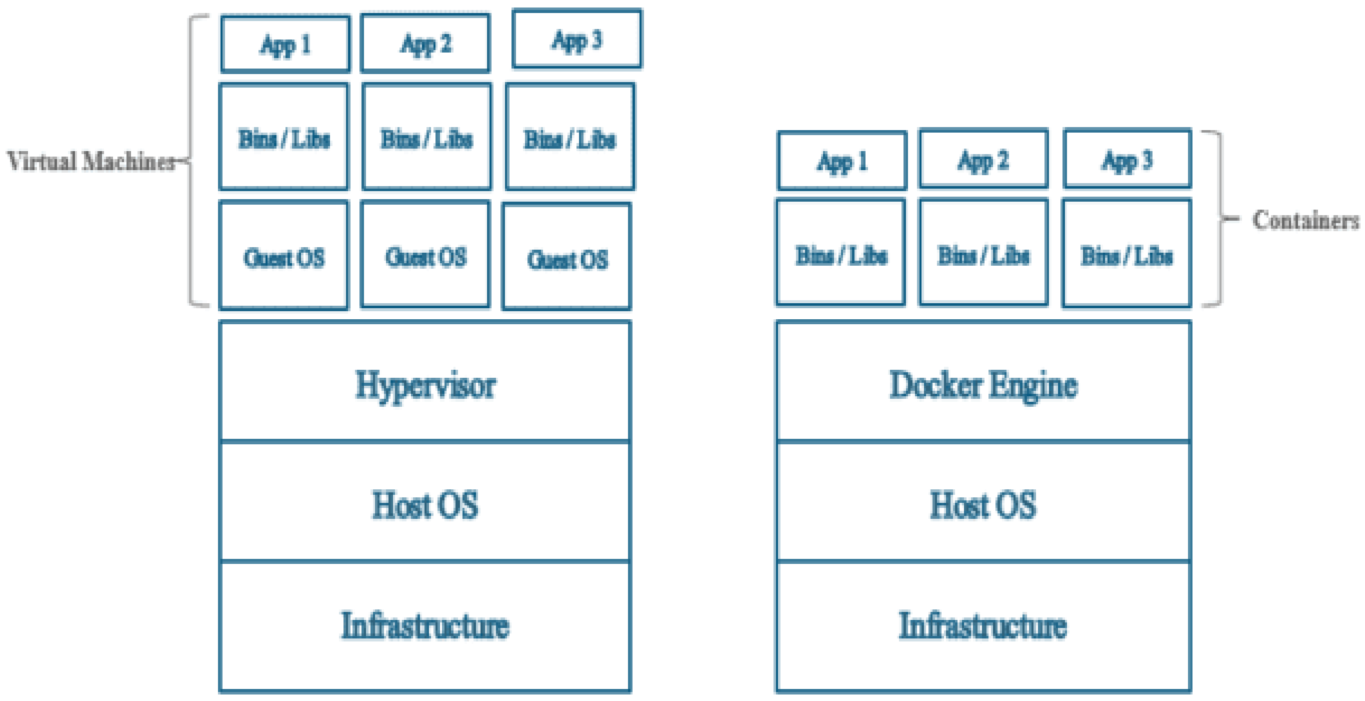
Docker

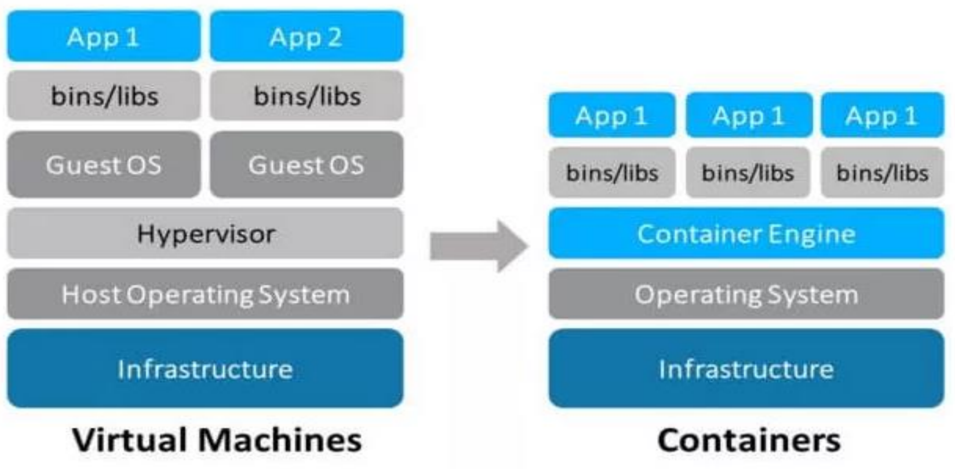
Container is like a VM, Docker's is a tool which create this VM Means, Containers, it is a deployment tool

Docker is Advanced Version of Virtualization.

Virtualization → Advance Version → **Docker**

Virtual Machine → Advance Version → **Container**





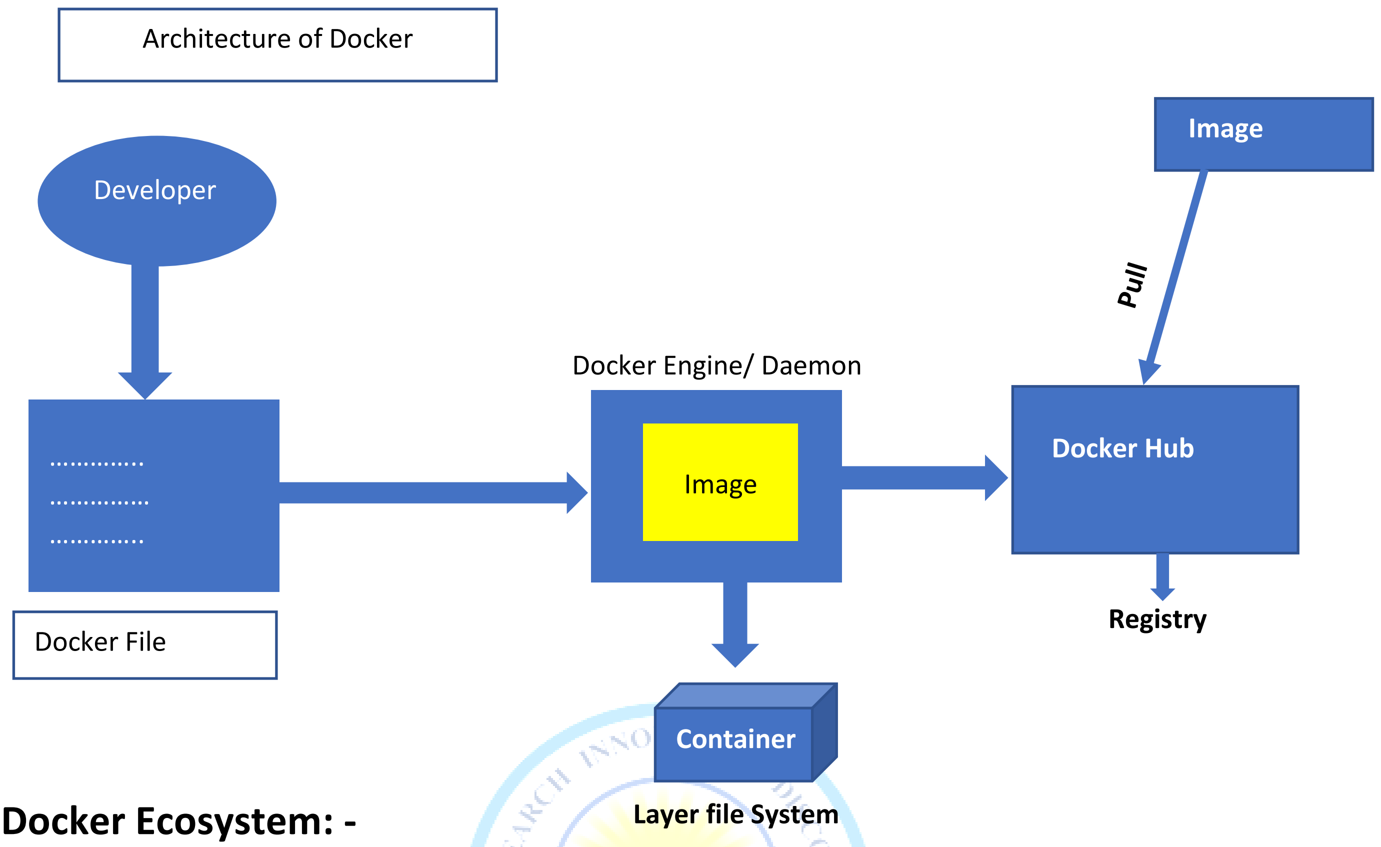
- Docker was first release in march 2013 it is developed by salmon and Sebastion pahl
- Docker is an open -source
- Centralised platform designed to "create deploy and run applications
- Docker uses container on the host O.S to run application it allows application to use the same Linux kernel as a system on the host computer, rather than creating a whole virtual O.S
- We can instay docker on any O.S to run application it allows applications to use the same linux kernel as a system on the host computer rather than creating a whole virtual O.S
- Docker Written in "Go" Language
- Docker is a tool is a tool that performs O.S level Virtualization also Known as containerization
- Before Docker many uses faces the problem that a particular code is running in the developer's . System
- Docker is O.S level Virtualization
- Docker is a set of "Platform as a service" that uses 0.S level Virtualization whereas VMware uses Hardware level virtualization .

Advantages of Docker

- No Pre-allocation of RAM
- CI Efficiency : - Docker enable you to build a container image and use that same image across every step of the deployment process
- Less cost
- It is light weight
- It can re-use the image
- It took very less time to create container.
- Running state of container is Image and image stipe is container

Disadvantage's of Docker

- Docker is not a good solution for application that required rich GUI.
- Default to manage large amount of container's
- Docker Does not provide cross platform compatibility means if an application is designed to run in a docker
- Difficult to manage large amount of container's
- Docker does not provide cross platform compatibility means if an application is designed to run in a docker
- Container on windows then it can't run on Linux or vice-versa
- Docker is suitable when the development o.s and testing o.s are same if the o.s is different we should
- No Solution for data recovery and Backup.



Docker Ecosystem: -

1. Docker Client
2. Docker Daemon or Server or Docker Engine
3. Docker Hub or Registry
4. Docker Images
5. Docker Compose

Components of Docker: -

1. Docker Daemon
   - Docker Daemon runs on the O.S it is responsible for running containers to manager's docker services.
   - Docker Daemon can communicate with other daemons.
2. Docker Client: -
   - Docker users can interact with docker daemon through a client (CLI)
   - Docker client uses commands and rest API to Communicate with the docker daemon
   - When a client runs any sever command on the docker client terminal the client terminal sends these docker
3. Docker Host
   - Docker Host is used to provide an environment to execute and run application it contains the docker daemon images, Containers networks and storages
4. Docker Hub/Registry: -
   - Docker registry manages and stores the docker images
   - There are two types of registries in the docker
   - Public Registry: - It is also called Docker hub.
   - Private Registry: - it is used share images within the enterprise.

5. Docker Images: -
   - Docker images are the read only binary templates used to create docker container's
   - Single File with all dependencies and Configuration required to run a program.
6. Docker Container's: -
   - Container's hold the entire packages that is needed to run the application or in the words we can say that the image is a template and the container is a copy of that template
   - Container's is like a VM.
   - Images becomes Container when they run on docker engine.

Way to Create an Images: -

I. Take Image from Docker Hub
II. Create image from docker file
III. Create image from existing docker containers

Docker Basic Command: -

1. To see all images, present in your local Machnine
   ```
   [...@...]# docker image
   ```
2. To find out images in docker hub
   ```
   [...@...]# docker  search Jenkins
   ```
3. To download image from docker hub to local machine
   ```
   [...@...]# docker pull Jenkins
   ```
4. To give name to container
   ```
   [...@...]# docker run it –name anjraj ubuntu/bin/bash
   ```
   (where i- interactive mode t-terminal)
5. To check service is start or not
   ```
   [...@...]# docker status
   ```
6. To start container
   ```
   [...@...]# docker start anjraj
   ```
   (it is container name)
7. To go inside container
   ```
   [...@...]# docker attach anjraj
   ```
8. To see all container's
   ```
   [...@...]# docker ps -a
   ```
9. To see only running container's
   ```
   [...@...]# docker ps
   ```
10. To stop container
    ```
    [...@...]# docker stop anjraj
    ```
11. To delete container
    ```
    [...@...]# docker rm anjraj
    ```

|| धन्यवाद ||

T3 Skill Center