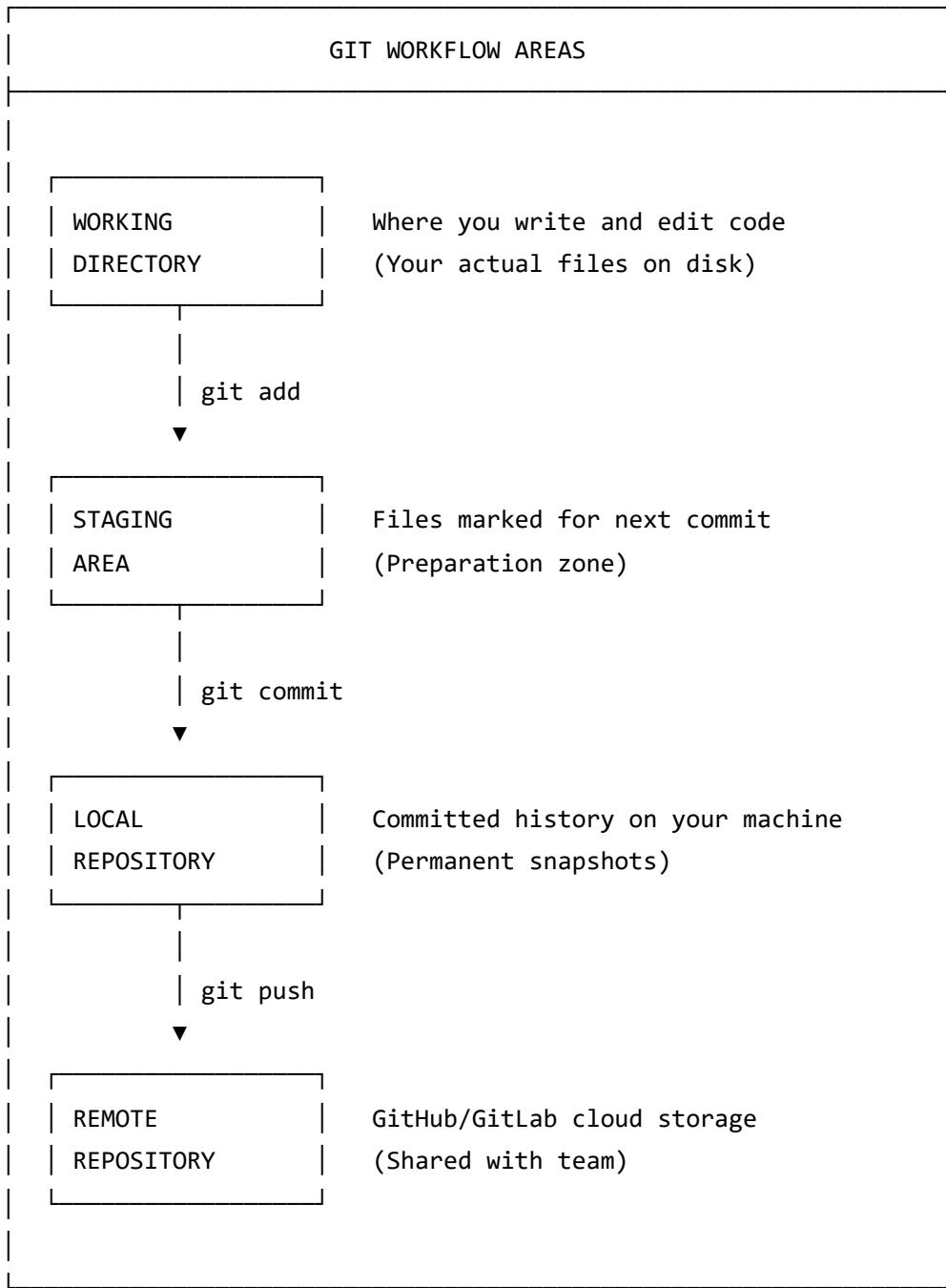


# **SECTIONS 4-6: Git Concepts, CI/CD Fundamentals & GitHub Actions**

# SECTION 4 — Git Concepts Explained Simply

## 4.1 The Three Areas of Git



## 4.2 Working Directory

### What It Is:

The folder on your computer where you edit files.

**Analogy:** Your desk where you're actively working on papers.

### **Key Points:**

- Contains actual files you can see and edit
- Changes here are NOT saved to Git yet
- If you delete a file, it's gone (unless committed)

### **Commands:**

```
# See what changed in working directory  
git status
```

```
# Discard changes in working directory  
git checkout -- filename.ts
```

## **4.3 Staging Area (Index)**

### **What It Is:**

A preparation zone for files you want to commit.

**Analogy:** An "Outbox" on your desk - papers you've reviewed and ready to file.

### **Why It Exists:**

- Pick specific files to commit
- Review changes before committing
- Create focused commits

### **Commands:**

```
# Add file to staging  
git add filename.ts
```

```
# Add all files  
git add .
```

```
# Remove from staging (keep changes in working directory)  
git reset filename.ts
```

```
# See what's staged  
git status
```

## 4.4 Commits

### What It Is:

A permanent snapshot of staged changes with a message.

**Analogy:** Making a photocopy and filing it. The original paper can change, but the copy is permanent.

### Key Points:

- Each commit has a unique ID (hash)
- Contains author, date, message, and changes
- You can always go back to any commit

### Commands:

```
# Create a commit  
git commit -m "Your message"  
  
# See commit history  
git log --oneline  
  
# See details of a commit  
git show <commit-hash>
```

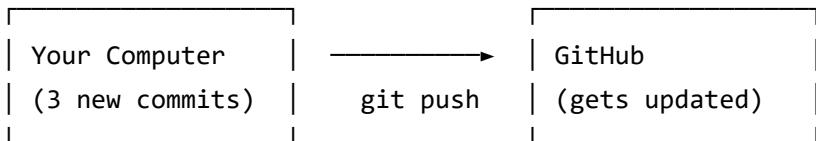
### Example Output:

```
a1b2c3d (HEAD -> feature/login) feat: add login tests  
d4e5f6g fix: update locator  
g7h8i9j refactor: simplify page object
```

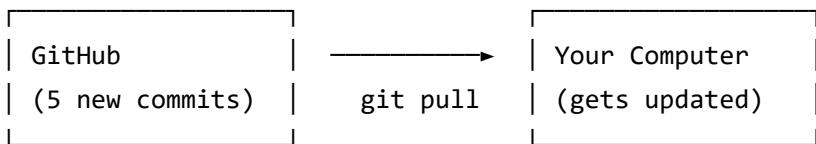
## 4.5 Push vs Pull

### Visual Comparison:

PUSH: Local → Remote (Upload)



PULL: Remote → Local (Download)



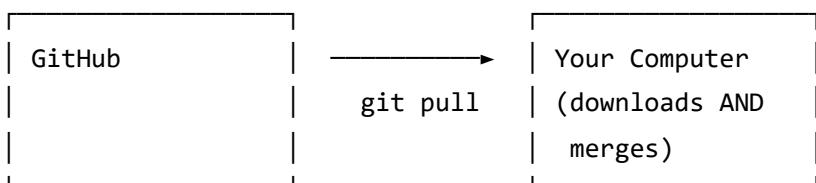
Aspect	Push	Pull
Direction	Local to Remote	Remote to Local
Purpose	Share your work	Get team's work
When	After committing	Before starting work
Command	<code>git push origin branch</code>	<code>git pull origin branch</code>

## 4.6 Fetch vs Pull

FETCH: Only downloads, doesn't merge



PULL: Downloads AND merges



Aspect	Fetch	Pull
Downloads changes	<input checked="" type="checkbox"/> Yes	<input checked="" type="checkbox"/> Yes

Aspect	Fetch	Pull
Merges changes	✗ No	✓ Yes
Safe	Safer	Can cause conflicts
Use case	Review first	Get latest quickly

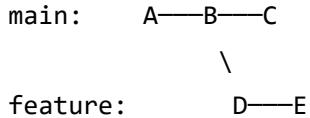
### When to Use Fetch:

```
git fetch origin
git log origin/develop    # See what's new
git merge origin/develop  # Then merge if happy
```

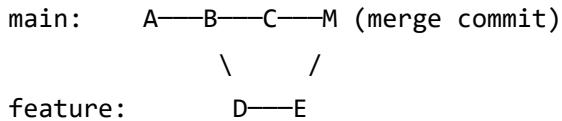
## 4.7 Merge vs Rebase

### Merge: Creates a merge commit

Before merge:

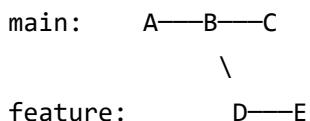


After merge:

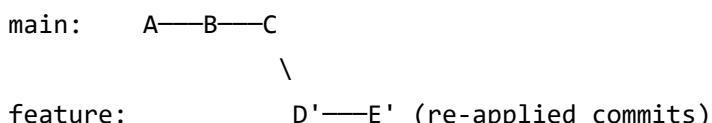


### Rebase: Moves commits to new base

Before rebase:



After rebase:



Aspect	Merge	Rebase
History	Preserves all branches	Linear history
Commit	Creates merge commit	No extra commits
Safety	Safer, non-destructive	Rewrites history
Team	Better for shared branches	Better for local branches
Command	<code>git merge branch</code>	<code>git rebase main</code>

### Enterprise Recommendation:

- Use **merge** for shared branches (develop, main)
- Use **rebase** for local feature branches before PR

## 4.8 Conflict Resolution

### What is a Conflict?

When Git can't automatically merge because the same lines were changed.

### When Conflicts Happen:

You:        Changed line 10 to "Hello"

Teammate: Changed line 10 to "Hi"

Git:        🤔 Which one to keep?

### Conflict Markers:

```
<<<<< HEAD
// Your version
const greeting = "Hello";
=====
// Their version
const greeting = "Hi";
>>>> feature/other-branch
```

### How to Resolve:

1. Find conflicted files:

```
git status      # Shows files with conflicts
```

## 2. Open file and decide which version to keep:

```
// After resolution - pick one or combine  
const greeting = "Hello World";
```

## 3. Remove conflict markers:

Delete the <<<<< , ====== , and >>>>> lines.

## 4. Stage and commit:

```
git add filename.ts  
git commit -m "fix: resolve merge conflict in greeting"
```

### Conflict Resolution Tips:

Tip	Description
Communicate	Talk to teammate who made the change
Understand both	Know why each change was made
Test after	Run tests after resolving
Small commits	Smaller PRs = fewer conflicts
Pull often	Frequent pulls reduce conflicts

## 4.9 Why Force Push is Dangerous

### What is Force Push?

```
git push --force    # ⚠ DANGEROUS  
git push -f        # ⚠ SAME THING
```

### Why It's Dangerous:

BEFORE your force push:

Remote: A—B—C—D—E (teammate's work)

Your: A—B—X—Y

AFTER your force push:

Remote: A—B—X—Y (D and E are GONE!)

## When Force Push Destroys Work:

1. Teammate pushed commits D and E
2. You didn't pull their changes
3. You force pushed your version
4. Their commits D and E are **permanently deleted**

## Safe Alternatives:

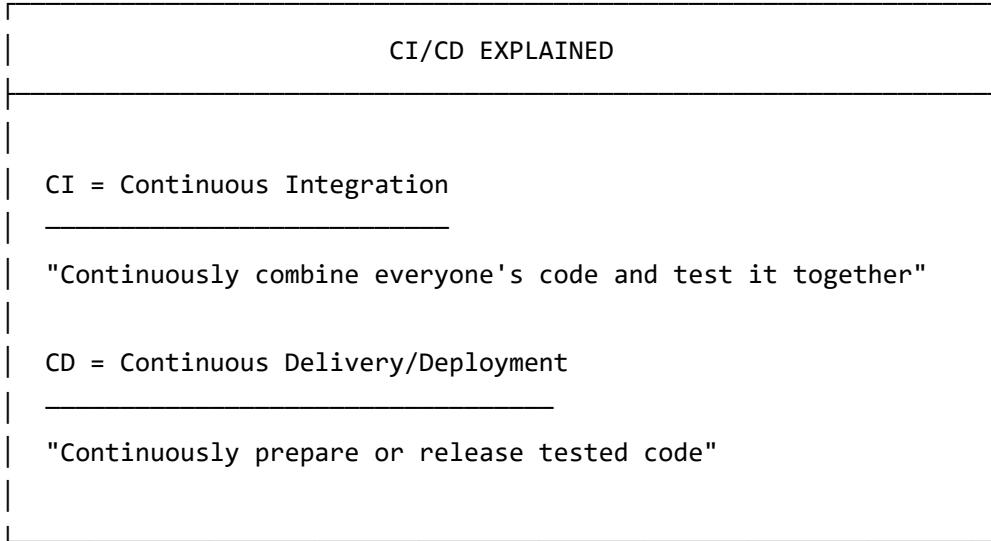
Instead of	Use	Why
<code>git push --force</code>	<code>git push --force-with-lease</code>	Fails if remote has new commits
Force pushing to shared branch	Merge or rebase properly	Don't rewrite shared history

## Enterprise Rule:

- 🚫 NEVER force push to `main` or `develop` branches
- ⚠ Only force push to your own feature branches when necessary

# SECTION 5 — CI/CD Fundamentals (Beginner Friendly)

## 5.1 What is CI/CD?



### Simple Analogy:

Concept	Analogy
CI	Restaurant kitchen automatically checking every dish before serving
CD	Automatically preparing dishes to be served when ready

## 5.2 Why Enterprises Need CI/CD

### Without CI/CD:

Developer A pushes code → Nobody tests it

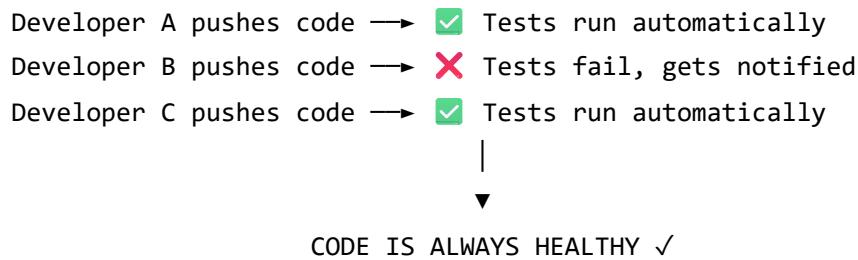
Developer B pushes code → Nobody tests it

Developer C pushes code → Nobody tests it



ONE MONTH LATER: Everything is broken 😱

### With CI/CD:



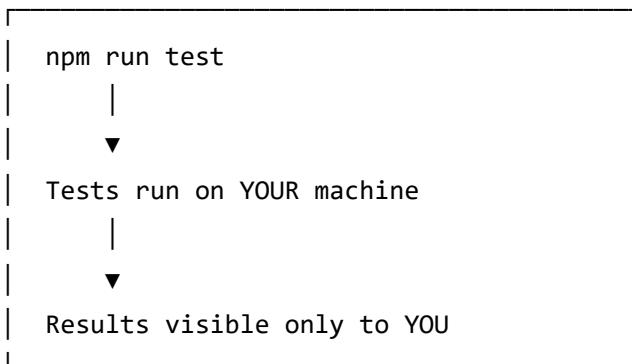
### Benefits for Enterprises:

Benefit	Description
Early Detection	Find bugs immediately, not after a month
Consistent Testing	Same tests run every time, in same environment
Time Savings	No manual test execution needed
Quality Assurance	Code can't merge if tests fail
Team Visibility	Everyone sees test results
Audit Trail	Complete history of all test runs

## 5.3 Local Execution vs CI Execution

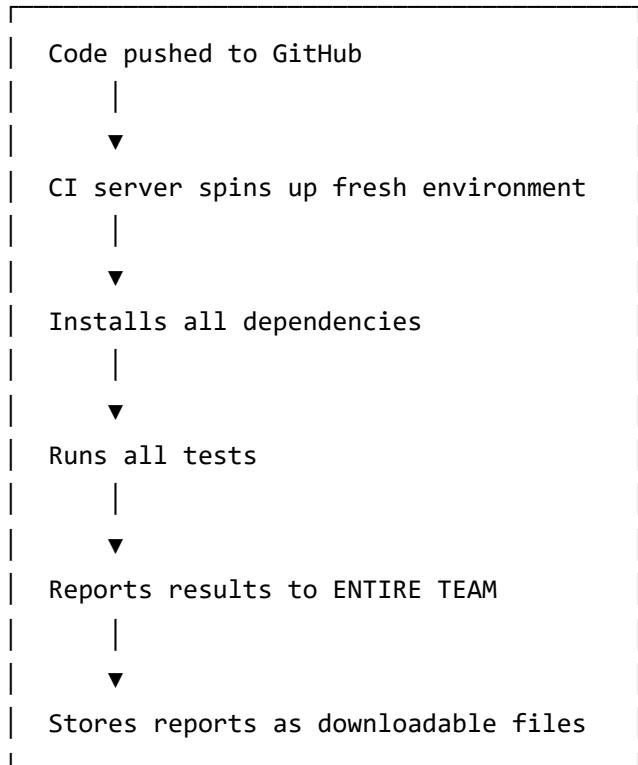
### Local Execution:

Your Computer



### CI Execution:

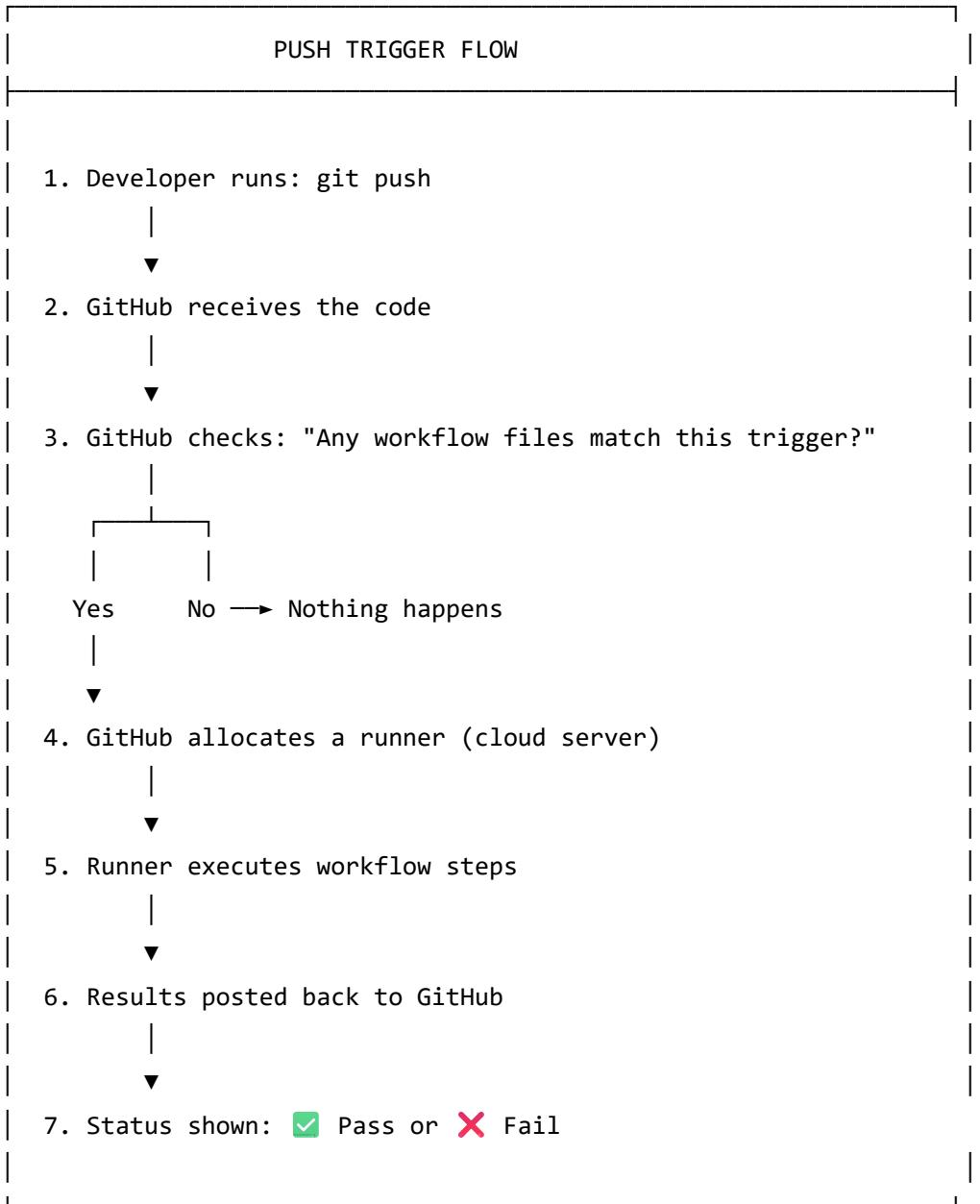
## GitHub Cloud Server



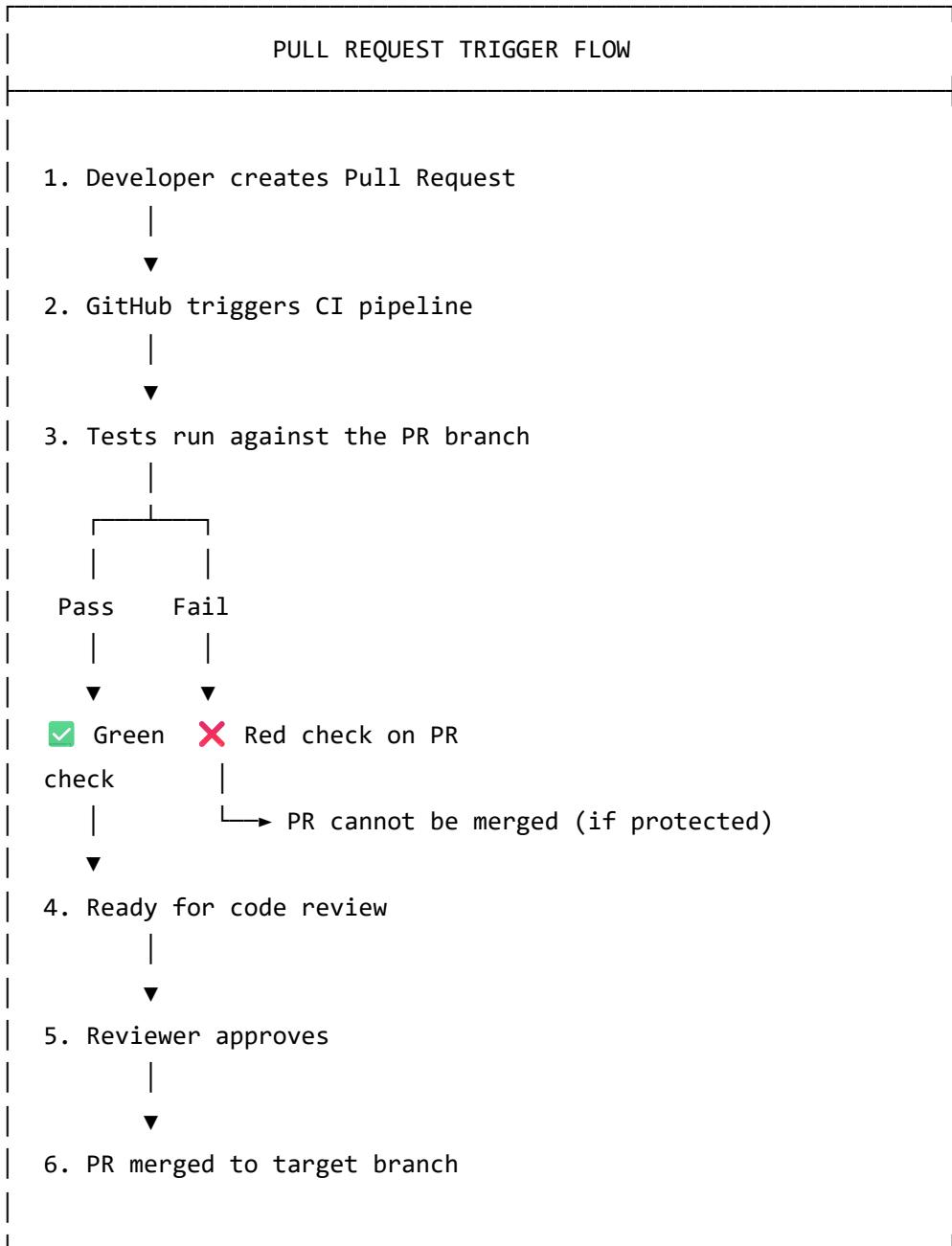
## Key Differences:

Aspect	Local	CI
Environment	Your machine	Fresh cloud server
Dependencies	Already installed	Installed fresh each time
Consistency	Varies by machine	Same every time
Visibility	Only you	Entire team
Reports	Local folder	Cloud artifacts
Speed	Faster (no setup)	Slower (includes setup)

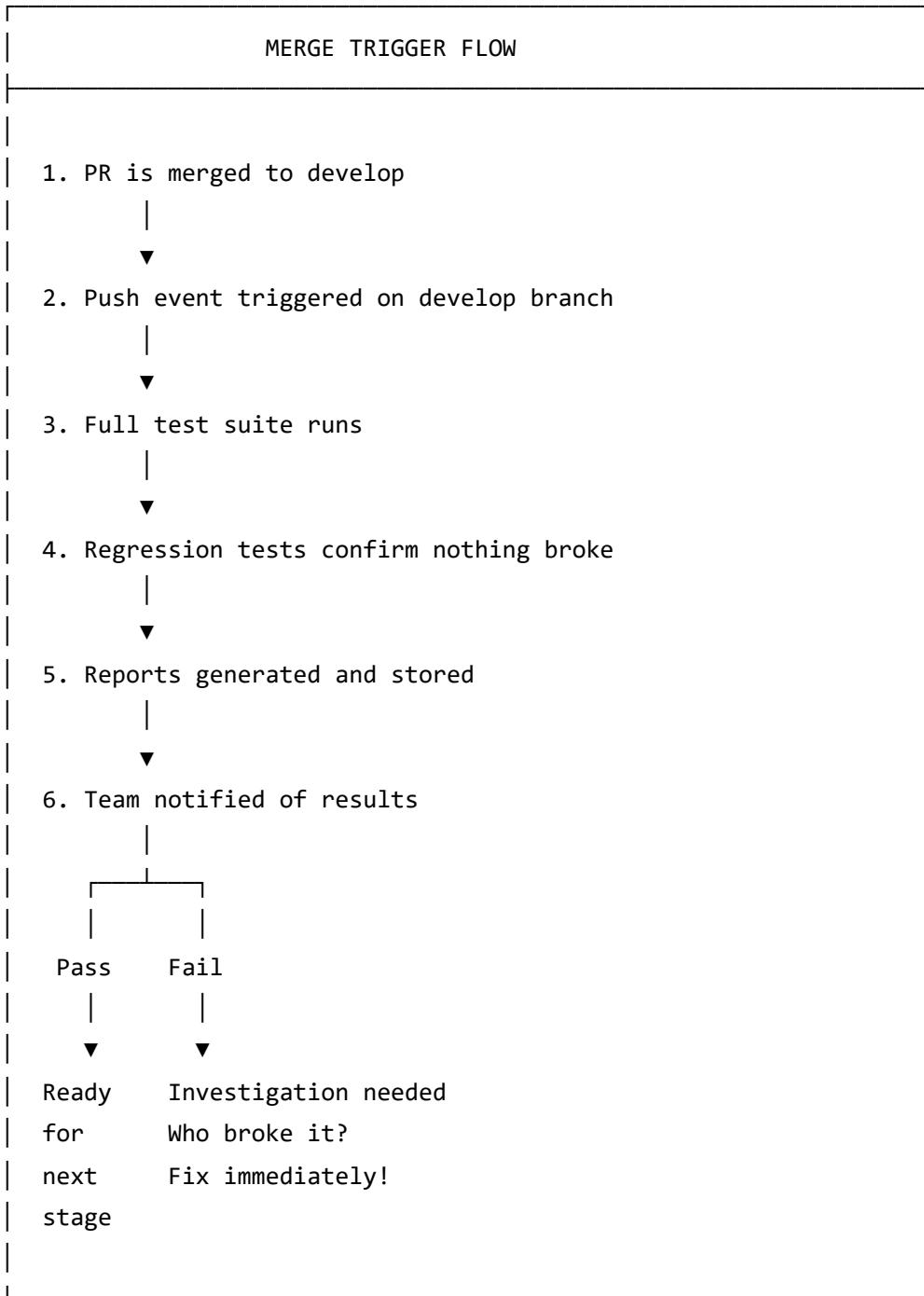
## 5.4 What Happens When Code is Pushed



## 5.5 What Happens When PR is Created



## 5.6 What Happens When PR is Merged



# SECTION 6 — GitHub Actions (Deep Dive)

## 6.1 What are GitHub Actions?

### Definition:

GitHub Actions is a CI/CD platform built directly into GitHub that allows you to automate workflows.

### Simple Explanation:

"GitHub Actions is like a robot assistant that watches your repository and automatically does tasks when certain events happen."

### Examples of Tasks:

- Run tests when code is pushed
- Build and deploy application
- Send notifications
- Generate reports
- Label issues automatically

## 6.2 Workflow File Structure

### Location:

```
your-repo/  
└── .github/  
    └── workflows/  
        └── playwright.yml    ← Workflow file
```

### Basic Structure:

```
name: Workflow Name          # Display name

on:                                # Triggers
  push:
    branches: [main]

jobs:                               # What to do
  job-name:
    runs-on: ubuntu-latest
    steps:
      - name: Step 1
        run: echo "Hello"
```

## 6.3 Complete Workflow Anatomy

```
# =====
# NAME: What appears in GitHub Actions tab
# =====
name: Playwright Tests

# =====
# ON: When should this workflow run?
# =====
on:
  push:
    branches: [main, develop]
  pull_request:
    branches: [main]
  workflow_dispatch:
  schedule:
    - cron: '0 0 * * *'

# =====
# JOBS: What work should be done?
# =====
jobs:
  test:
    name: Run Tests
    runs-on: ubuntu-latest
    timeout-minutes: 60

# =====
# STEPS: Individual tasks in sequence
# =====
steps:
  - name: Checkout code
    uses: actions/checkout@v4

  - name: Setup Node.js
    uses: actions/setup-node@v4
    with:
      node-version: 20

  - name: Install dependencies
    run: npm ci
```

```

- name: Run tests
  run: npx playwright test
  env:
    BASE_URL: ${ secrets.BASE_URL }

- name: Upload report
  uses: actions/upload-artifact@v4
  if: always()
  with:
    name: playwright-report
    path: playwright-report/

```

## 6.4 Jobs Explained

### What is a Job?

A job is a set of steps that execute on the same runner (server).

```

jobs:
  # Job 1: Run tests
  test:
    runs-on: ubuntu-latest
    steps:
      - run: npx playwright test

  # Job 2: Deploy (only if tests pass)
  deploy:
    needs: test           # Wait for 'test' job
    runs-on: ubuntu-latest
    steps:
      - run: npm run deploy

```

### Job Properties:

Property	Description	Example
runs-on	Operating system	ubuntu-latest , windows-latest
timeout-minutes	Max job duration	60
needs	Job dependencies	needs: test
if	Conditional execution	if: success()

Property	Description	Example
env	Environment variables	env: MY_VAR: value
strategy	Matrix configuration	Browser matrix

## 6.5 Steps Explained

### What is a Step?

A single task within a job. Steps run in sequence.

### Two Types of Steps:

steps:

```
# TYPE 1: Action (uses)
- name: Checkout code
  uses: actions/checkout@v4

# TYPE 2: Command (run)
- name: Run tests
  run: npx playwright test
```

### Step Properties:

Property	Description	Example
name	Display name	Install dependencies
uses	Action to use	actions/checkout@v4
run	Command to execute	npm install
with	Action inputs	node-version: 20
env	Environment variables	CI: true
if	Conditional	if: failure()
continue-on-error	Don't fail job	true

## 6.6 Runners Explained

### What is a Runner?

A server (virtual machine) that executes your workflow.

### Types of Runners:

Type	Description	Cost
GitHub-hosted	Managed by GitHub	Free (limits apply)
Self-hosted	Your own servers	Your infrastructure

### Available GitHub-Hosted Runners:

Runner	OS	Use Case
ubuntu-latest	Linux	Most common, fastest
windows-latest	Windows	Windows-specific tests
macos-latest	macOS	Safari/iOS testing

## 6.7 Secrets Explained

### What are Secrets?

Encrypted environment variables for sensitive data.

### How to Create Secrets:

1. Go to repository → Settings
2. Click Secrets and variables → Actions
3. Click "New repository secret"
4. Add name and value

### Using Secrets in Workflow:

```

steps:
  - name: Run tests
    run: npx playwright test
    env:
      TEST_USERNAME: ${{ secrets.TEST_USERNAME }}
      TEST_PASSWORD: ${{ secrets.TEST_PASSWORD }}
      API_KEY: ${{ secrets.API_KEY }}

```

## Secret Levels:

Level	Scope	Use Case
Repository	Single repo	Repo-specific credentials
Organization	All repos in org	Shared credentials
Environment	Specific env	Dev vs Prod secrets

## 6.8 Environment Variables

### Two Ways to Set:

```

# Level 1: Workflow level
env:
  NODE_ENV: test

jobs:
  test:
    # Level 2: Job level
    env:
      DEBUG: pw:api

steps:
  - name: Run tests
    # Level 3: Step level
    env:
      HEADLESS: true
    run: npx playwright test

```

### Built-in Variables:

Variable	Description
<code>\${{ github.repository }}</code>	Owner/repo name
<code>\${{ github.ref }}</code>	Branch/tag ref
<code>\${{ github.sha }}</code>	Commit SHA
<code>\${{ github.actor }}</code>	Username who triggered
<code>\${{ github.run_id }}</code>	Unique run ID

## 6.9 Triggers Deep Dive

### Push Trigger

```
on:
  push:
    branches:
      - main
      - develop
      - 'feature/**'          # Wildcard: all feature branches
    paths:
      - 'tests/**'            # Only if tests changed
      - '!docs/**'            # Ignore docs changes
```

### Pull Request Trigger

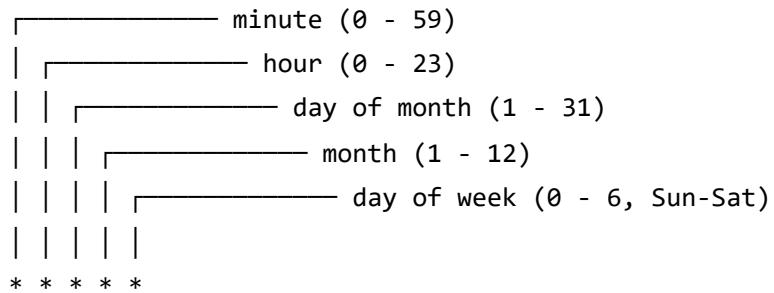
```
on:
  pull_request:
    branches:
      - main
    types:                      # Types of PR events
      - opened
      - synchronize           # New commits pushed
      - reopened
```

# Schedule Trigger (Cron)

on:

```
schedule:  
  - cron: '0 0 * * *'      # Daily at midnight UTC  
  - cron: '0 9 * * 1'       # Every Monday at 9 AM UTC
```

## Cron Syntax:



Schedule	Cron	Description
Daily midnight	0 0 * * *	Every day at 00:00 UTC
Weekdays 9 AM	0 9 * * 1-5	Mon-Fri at 9 AM UTC
Hourly	0 * * * *	Every hour
Weekly Monday	0 0 * * 1	Every Monday midnight

## Manual Trigger (workflow\_dispatch)

```
on:
  workflow_dispatch:
    inputs:
      environment:
        description: 'Environment to test'
        required: true
        default: 'staging'
        type: choice
        options:
          - development
          - staging
          - production
      browser:
        description: 'Browser to use'
        required: false
        default: 'chromium'
```

### Using Input Values:

```
steps:
  - name: Run tests
    run: npx playwright test --project=${{ inputs.browser }}
```

## 6.10 YAML Keywords Summary

Keyword	Purpose	Example
name	Display name	name: CI Pipeline
on	Triggers	on: push
jobs	Job definitions	jobs: test:
runs-on	Runner OS	runs-on: ubuntu-latest
steps	Job steps	steps: - run: test
uses	Use an action	uses: actions/checkout@v4
run	Run command	run: npm test

Keyword	Purpose	Example
with	Action inputs	with: node-version: 20
env	Environment vars	env: CI: true
if	Conditional	if: failure()
needs	Job dependency	needs: build
strategy	Matrix/parallel	strategy: matrix:
secrets	Access secrets	secrets.TOKEN

*Continue to Section 7-9...*