Motilal Nehru National Institute of Technology Allahabad

Software Testing
Program : MCA V Semester
Tool : GIT

Presented By:

Anubhav Krishna (2023CA20)
Utkarsh Dubey (2023CA109)
Yashika Jain (2023CA116)

Course Coordinator:

Dr. Anoj Kumar,
Associate Professor(CSED)

Roles of Team Members

Roles and Responsibilities:

- **Utkarsh Dubey** – Project Coordinator,Tool Setup and Benchmark testing
- **Yashika Jain** – Literature Review,Tool Survey and Test Case Writing
- **Anubhav Krishna** – Results, Analysis, and Documentation

- All members contributed to research, implementation, and slide content.
- Work was divided equally for smooth and timely project completion.

Introduction of GIT Testing Tool

- Git Test Tools are utilities and frameworks used to test workflows, hooks, and integrations in Git-based development environments.
- These tools help ensure that Git operations (like commits, merges, and pull requests) follow expected behavior and project standards.
- Popular tools like Prettier, Husky, and Lint-Staged can be integrated into Git hooks to run checks automatically during version control operations.

Why Use GIT?

- Automates quality checks before code is committed or pushed.
- Ensures consistency in code formatting, style, and behavior across the team.
- Prevents broken code or non-compliant commits from entering the main branch.

Category: *Version Control Testing – Static & Workflow Enforcement*

Literature Survey on GIT Tool

Git testing tools have received significant attention in the software engineering community for enhancing code quality and enforcing workflow standards. These tools are often discussed in blogs, GitHub repositories, and DevOps-focused documentation due to their critical role in maintaining reliable version control processes.

Key Contributions & Insights:

- **Git Docs & Hooks** – Official documentation details how Git hooks can automate testing, linting, and validation tasks.
- **Husky** – Popular tool for managing Git hooks, enabling pre-commit/push scripts to catch issues early.
- **Community & Case Studies** – Blogs and forums (e.g., Medium, Dev.to) showcase practical Git testing workflows using tools like Husky and Danger.js.
- **Industry Adoption** - Companies like Microsoft, Shopify, and GitHub use these tools to ensure clean, test-verified code commits in large-scale projects.

Step-by-Step Working of GIT (Part 1)

Setup & Trigger Initiation

1. **Write Hook Scripts** Create scripts to run on Git events like pre-commit, pre-push, or commit-msg.
2. **Configure Git Hooks** Use the .git/hooks directory manually or tools like Husky to automate setup.
3. **Integrate with Tools** Binds scripts to Git lifecycle events.

Step-by-Step Working of GIT (Part 2)

Execution & Validation:

4. **Hook Triggered Automatically:** When a Git event occurs (e.g., commit), the corresponding hook runs configured scripts.
5. **Run Checks or Tests:** Scripts may include: Code linters,Unit/integration tests,Formatting checks,Security scans.
6. **Feedback & Enforcement:**
    - If a script fails, the Git action is blocked (e.g., commit rejected).
    - Ensures only clean, tested code enters the repository.

Tool Status: GIT

Current Status of GIT Tool:

- **Actively Maintained:** Tools like Husky, Lint-Staged, and Danger.js are frequently updated and supported by a vibrant open-source community.
- **Extensively Used:** Widely adopted across industries to enforce code quality, especially in CI/CD pipelines and collaborative development.
- **Ecosystem Friendly:** Seamlessly integrates with Git, npm, Prettier, ESLint, and CI platforms like GitHub Actions and GitLab CI.
- **Thriving Community:** Strong community presence on GitHub, Dev.to, and Medium, with ample tutorials, plugins, and contributions.
- **Open Source:** Most Git testing tools are open-source under permissive licenses (e.g., MIT), encouraging transparency and team-wide adoption.

Software and Hardware Specifications

Software Requirements:

- **Git:** Version 2.30 or above (latest stable recommended)
- **Node.js:** Version 14 or higher (for tools like Husky, Lint-Staged, etc.)
- **Text Editor/IDE:** VS Code, WebStorm, or any Git-integrated editor
- **Operating System:** Windows, macOS, or Linux
- **Package Managers:** npm or yarn (for installing Git hook tools)
- **CI/CD Platform (Optional):** GitHub Actions, GitLab CI, Jenkins, etc.

Hardware Requirements:

- **RAM:** Minimum 2 GB (4 GB or more recommended for multitasking with Git tools)
- **Disk Space:** At least 500 MB of free space
- **Processor:** Dual-core or better for smooth performance in development workflows

Note: Git and related testing tools are lightweight and run efficiently on most standard development setups.

Experimental Setup & Test Scenarios

Tools & Environment:

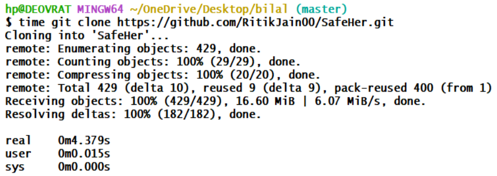
- **Version Control:** Git v2.43.0
- **Node Version:** v20.12.1
- **Package Manager:** npm v10.5.0
- **Hook Manager:** Husky v9.0.0 with Lint-Staged

Test Scenarios:

- **Pre-commit Hook Testing:**
    - Lint staged files using ESLint, block commit on error
- **Pre-push Hook Testing:**
    - Run unit tests before pushing, simulate failed push on test error
- **Custom Hook Validation:**
    - Enforce commit message format, e.g., issue ID presence

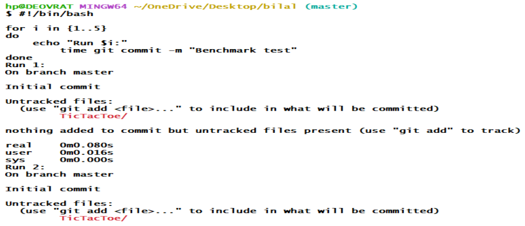
Execution Results and Observations

```
//time git clone
```



```
hp@DEOVRAT MINGW64 ~/OneDrive/Desktop/bilal (master)
$ time git clone https://github.com/RitikJain00/SafeHer.git
Cloning into 'SafeHer'...
remote: Enumerating objects: 429, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (20/20), done.
remote: Total 429 (delta 10), reused 9 (delta 9), pack-reused 400 (from 1)
Receiving objects: 100% (429/429), 16.60 MiB | 6.07 MiB/s, done.
Resolving deltas: 100% (182/182), done.

real    0m4.379s
user    0m0.015s
sys     0m0.000s
```

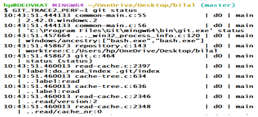
```
//using script loop for time difference
```



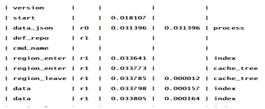
```
hp@DEOVRAT MINGW64 ~/OneDrive/Desktop/bilal (master)
$ #!/bin/bash

for i in {1..5}
do
    echo "Run $i:"
        time git commit -m "Benchmark test"
done
Run 1:
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        TicTacToe/

nothing added to commit but untracked files present (use "git add" to track)

real    0m0.080s
user    0m0.016s
sys     0m0.000s
Run 2:
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        TicTacToe/
```

Execution Results and Observations (Cont.)

//git trace....

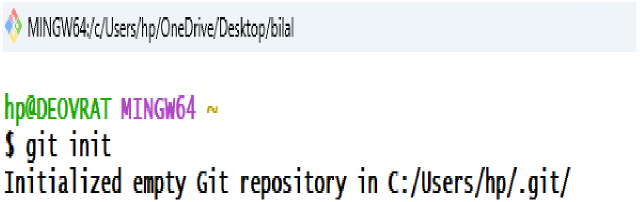


```
hp@DEOVRAT MINGW64 ~/OneDrive/Desktop/bilal (master)
$ GIT_TRACE2_PERF=1 git status
10:43:51.444133 common-main.c:55             | d0 | main
    | 2.42.0.windows.2
10:43:51.444133 common-main.c:56             | d0 | main
    | 'C:\Program Files\Git\mingw64\bin\git.exe' status
10:43:51.457664 ..._win32_process_info.c:120 | d0 | main
    | windows/ancestry:["bash.exe","bash.exe"]
10:43:51.458673 repository.c:143             | d0 | main
    | worktree:C:/Users/hp/OneDrive/Desktop/bilal
10:43:51.458673 git.c:464                    | d0 | main
    | status (status)
10:43:51.460013 read-cache.c:2397            | d0 | main
    | label:do_read_index .git/index
10:43:51.460013 cache-tree.c:634             | d0 | main
    | ..label:read
10:43:51.460013 cache-tree.c:636             | d0 | main
    | ..label:read
10:43:51.460013 read-cache.c:2346            | d0 | main
    | ..read/version:2
10:43:51.460013 read-cache.c:2348            | d0 | main
    | ..read/cache_nr:0
```



```
| version      |    |          |          |
| start        |    | 0.018107 |          |
| data_json    | r0 | 0.031396 | 0.031396 | process
| def_repo     | r1 |          |          |
| cmd_name     |    |          |          |
| region_enter | r1 | 0.033641 |          | index
| region_enter | r1 | 0.033773 |          | cache_tree
| region_leave | r1 | 0.033785 | 0.000012 | cache_tree
| data         | r1 | 0.033798 | 0.000157 | index
| data         | r1 | 0.033805 | 0.000164 | index
```

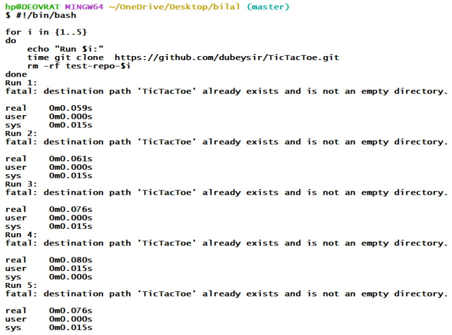
//git init



```
MINGW64:/c/Users/hp/OneDrive/Desktop/bilal

hp@DEOVRAT MINGW64 ~
$ git init
Initialized empty Git repository in C:/Users/hp/.git/
```

Execution Results and Observations (Cont.)

//git clone for 5 run



```
hp@DEOVRAT MINGW64 ~/OneDrive/Desktop/bilal (master)
$ #!/bin/bash

for i in {1..5}
do
    echo "Run $i:"
    time git clone  https://github.com/dubeysir/TicTacToe.git
    rm -rf test-repo-$i
done
Run 1:
fatal: destination path 'TicTacToe' already exists and is not an empty directory.

real    0m0.059s
user    0m0.000s
sys     0m0.015s
Run 2:
fatal: destination path 'TicTacToe' already exists and is not an empty directory.

real    0m0.061s
user    0m0.000s
sys     0m0.015s
Run 3:
fatal: destination path 'TicTacToe' already exists and is not an empty directory.

real    0m0.076s
user    0m0.000s
sys     0m0.015s
Run 4:
fatal: destination path 'TicTacToe' already exists and is not an empty directory.

real    0m0.080s
user    0m0.015s
sys     0m0.000s
Run 5:
fatal: destination path 'TicTacToe' already exists and is not an empty directory.

real    0m0.076s
user    0m0.000s
sys     0m0.015s
```

Git Hooks Tool Comparison: Execution Usage

Tool	**Use Case**	**Ease of Setup**	**Performance**	**Remarks**
Husky	Pre-commit/push hooks	Easy	Fast	Popular for JS projects. Integrates well with lint-staged.
lint-staged	Run linters on staged files	Medium	Very Fast	Works with Husky to optimize commit-time checks.
Danger.js	Code review automation	Medium	Fast	Great for CI, custom messages in PRs.
Custom Git Hooks	Fully customizable scripts	Hard	Varies	Powerful but requires manual setup and maintenance.

Limitations of Git Testing Tools

While helpful in improving code quality, Git testing tools also have some limitations:

- **Limited to Git Workflows:** Only applicable in Git-based projects, reducing usability in other version control systems.
- **Not a Full Testing Solution:** Primarily assists in pre-commit/pre-push checks; not a replacement for unit or E2E testing frameworks.
- **Initial Setup Complexity:** Tools like Husky and Danger.js can be challenging to configure for newcomers.
- **Performance Bottlenecks:** Running linters or scripts on large staged files may slow down the commit/push process.
- **Bypass Risk:** Developers can skip hooks (e.g., using `--no-verify`), potentially pushing unverified code.

Conclusion

Summary of Git-Based Testing Tools:

- Git hooks enable automated testing and code checks during development workflows.
- Tools like Husky and Lint-Staged integrate easily with Git for pre-commit and pre-push validations.
- Helps enforce code standards and reduce bugs before deployment.

Key Takeaways:

- Enhances code quality through early-stage validations.
- Seamlessly integrates with CI/CD pipelines.
- Supported by a wide range of plugins and community tools.
- Widely adopted by industry leaders like Microsoft, GitHub, and Shopify.

References

Key Resources for Further Learning:

- **Git Documentation:** https://git-scm.com/docs/githooks
- **Husky GitHub Repository:** https://github.com/typicode/husky
- **Lint-Staged GitHub Repository:** https://github.com/okonet/lint-staged
- **Dev.to & Medium Blogs:** Tutorials and use cases for Git hooks in real-world projects.
- **YouTube Tutorials:** Git Hooks Husky Video Guides