```
Usage:

  gitclick create [<repository>] [on <account>] [--no-issues] [--no-wiki] [--private]

    Creates a public repo[illegible] with issues and wiki by defa[illegible]
    <repository> defau[illegible] of the cur[illegible]
    <account> defaul[illegible] account

  gitclick list [illegible] existi[illegible]
  gitclick use <accou[illegible] account [illegible]
  gitclick add [illegible] active pr[illegible] for [illegible] ting a new account
  gitclick remove <account> [illegible] emove <account>

  gitclick -v, --version       Output version number
  gitclick -h, --help          Output usage information

$
```

Content

Section 1

Understanding Git Fundamentals

Introduction to Version Control with Git

01

What is Version Control?

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later. It is essential for tracking the evolution of code and collaborating effectively.

02

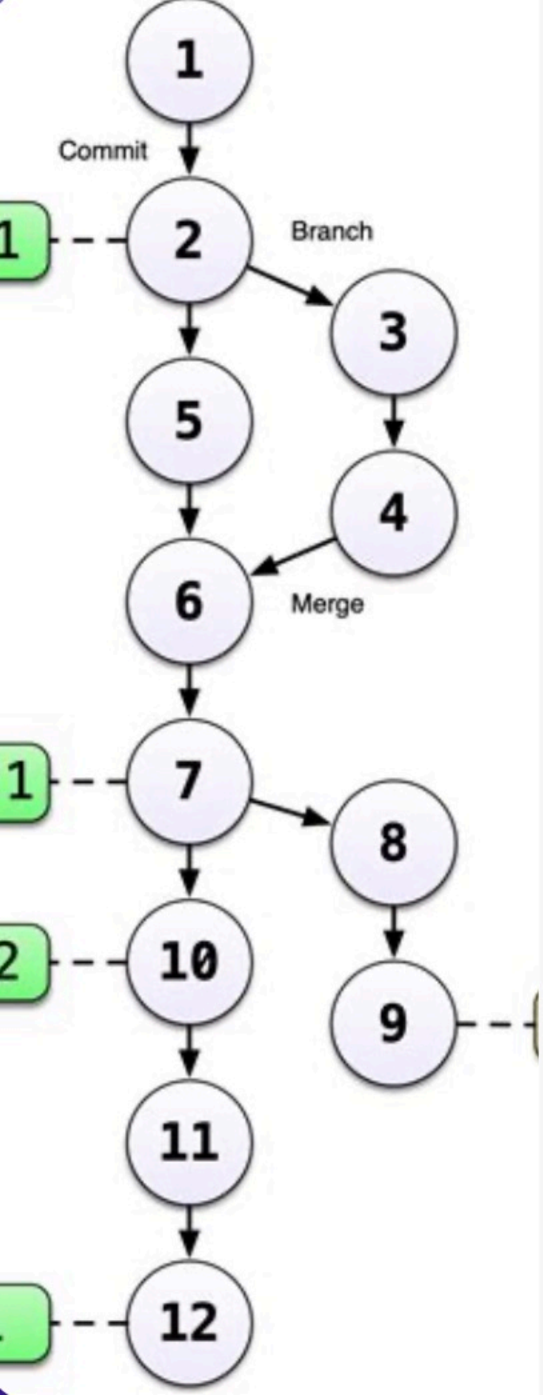
Why Use Git?

Git is a distributed version control system that provides a robust and flexible platform for managing changes to your codebase. It enables students to work on projects collaboratively and maintain a history of their work.

03

Key Benefits of Git

Git enables collaboration, facilitates experimentation, and ensures the integrity and traceability of project history, fostering a culture of teamwork and accountability in software development.



Exploring Git Stash and Reflog

Git Stash

Temporarily shelves changes in your working directory so you can switch branches or work on something else without committing your changes. Useful for saving work in progress or temporarily hiding changes.

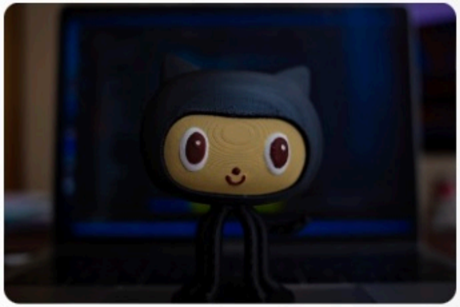
Git Reflog

Records the history of actions performed on your repository, including commits, merges, rebases, resets, etc. Useful for recovering lost commits, undoing accidental changes, or navigating through complex histories.

Utilizing Stash and Reflog

Understanding these features equips students with the ability to manage their work-in-progress effectively and troubleshoot potential issues in their projects.

Utilizing Git Diff and Switch

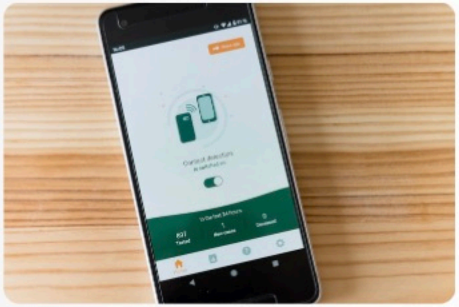


Git Diff

Shows the difference between changes in files, commits, or branches. Useful for reviewing changes before committing, comparing branches, or understanding modifications made over time.

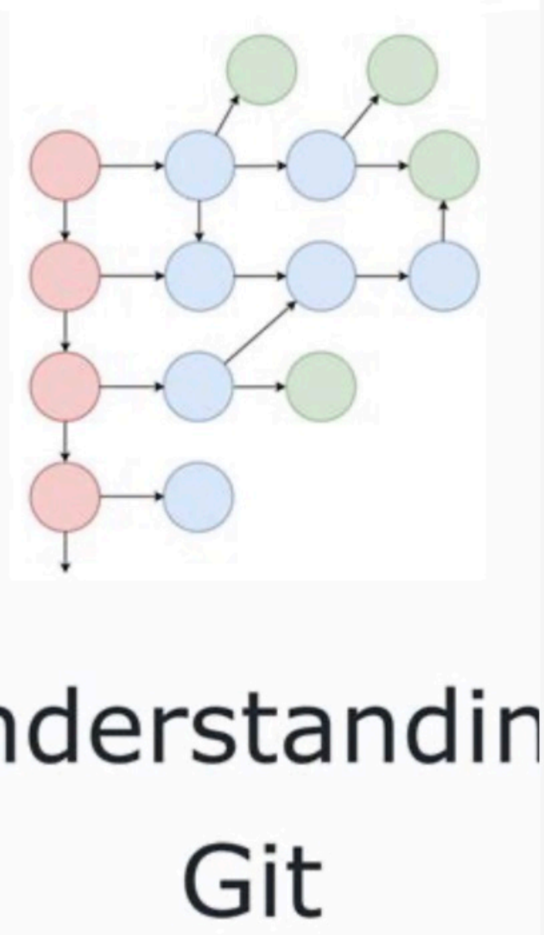
Git Switch

The git switch command is used to switch branches, enabling students to navigate between different project states and work on distinct features or fixes seamlessly.



Practical Application

These commands can be utilised to compare changes, understand branching, and manage their project's codebase effectively.



Understanding the Basics of Git Rebase

What is Git Rebase?

Rewrites commit history by moving, combining, or deleting commits. Useful for maintaining a clean and linear history by incorporating changes from one branch into another.

When to Use Git Rebase

Git rebase is useful for maintaining a clean, linear project history by rewriting the commit history, providing people with insights into managing project history effectively.

Practical Implications

Git rebase is commonly used to integrate feature branches cleanly, squash multiple commits into one for clarity, and resolve merge conflicts efficiently by applying commits individually.

Section 2

Advanced Git Techniques

Exploring Git Bisect

Understanding Git Bisect

Git bisect is a powerful tool for finding the commit that introduced a bug by using a binary search algorithm, enabling students to troubleshoot and identify issues in their codebase effectively.

How to Use Git Bisect

By marking known 'good' and 'bad' commits, Git bisect guides students through a process of elimination to identify the problematic commit, enhancing their debugging skills.

Real-world Application

Git Bisect command aids developers in identifying and isolating the exact change responsible for the bug, facilitating targeted debugging and resolution efforts.

ssing the power of tech

n), Oil and Natural Gas Corporation (ONGC)

Throughout the history of the oil and gas industry the tools and techniques of exploration have been continually improving, both in performance and economy. This progress has been in response to an unrelenting pressure to develop new capabilities after existing ones have become inadequate to find new hydrocarbon reservoirs. In addition to the areas newly opened for exploration, most geophysical surveys are undertaken where previous ones have failed because the instrument, field technique, processing or interpretational methods were not able to image sub-surface to the desired precision. Reservoirs that can be located with the existing technology are the only ones that will be discovered and the remainder will not be found until the technology improves sufficiently. Thus, the exploration geoscientist finds himself on an accelerating treadmill and must run faster and faster just to stay where he is.

ONGC's Creation – the dawn of an era of oil Exploration & Production in India

Back in 1955, the government of India decided that exploration and production of hydrocarbons would be taken up indigenously and made it a national policy with the exclusive responsibility of the state. Western countries were of the opinion that, with the exception of Assam, there was no possibility of hydrocarbon habitats in the rest of India. They were neither ready to sell the equipment required for hydrocarbon exploration, nor to train Indian scientists/engineers in this domain. India at that point in time was following a socialistic dream and wanted self sufficiency in hydrocarbons, both technically as well as economically. Naturally, India turned to the Soviet Union for help and support. The Soviets, knowing the importance of hydrocarbons in the geopolitical scenario, wanted a foothold in the Middle East and Southeast Asia. Thus, demand for hydrocarbons in India and the Soviet block's ambitions became complementary to one another and led to a very successful era of cooperation between the two countries. In 1956 the Geological Survey of India, Directorate of Oil and Gas was transformed into the Oil and Natural Gas Commission (ONGC) by an act of Parliament. Geologists, geo-physicists, and drilling engineers were recruited to start ONGC with its headquarters at Dehradun and Sh. K D Malaviya as its chairman. Geophysical exploration initiated by

the GSI was intensified by t
geophysicists, and on the ba
studies a number of locat
drilling in Gujarat and Assa

In Exploration, nothing s

With highly encouraging
well, Lunej-1, and subsequ
Gujarat and Rudrasagar in
a strategy to increase exp
the prospective sedimenta
activities kept pace with the
in seismic technology work
developed a high degree of
of exploration, both onland

In the present exploration
resource to in-place oil is l
challenging endeavour w
nearing an end. In future, l
to be found in deep wat
logistically difficult areas,
production more difficult,
greater economic risk.

The success of exploratio
the power of technology.
and exploitation heavily
data acquisition techniqu
operations, graphics and vi
translating various data an
models for realistic visualis

Cutting Edge Technologi

Geophysical prospecting is
exploration value chain, w
bring about a substantial
accretion and reduce explor
industry has made great
significantly to the growth o
Recent developments in ge
revolutionised exploration
about far-reaching qualita
practices. The following is
the key technologies consid
the risk of exploration. A
been implemented in Indi
stages of implementation.

Q- Marine

This technology, using a si
aims to improve the hand-

FIRST

Harnessing the Power of Git Cherry-Pick

Git Cherry-Pick

Selectively applies a single commit from one branch to another. Useful for bringing in specific changes from one branch without merging the entire branch, especially for hotfixes or applying changes across different branches.

Practical Use Cases

Git cherry-pick is useful for selectively applying specific commits to different branches, demonstrating its relevance in managing project features and fixes.

Application in Project Management

We can leverage this technique to integrate specific changes into their project branches, fostering a deeper understanding of version control.

Leveraging Git Reflog for Troubleshooting

01

Git Reflog in Troubleshooting

The reflog is a helpful tool for recovering lost commits or branches, and for undoing complex operations, providing us with a safety net for managing their project history.

02

Recovering Lost Work

The reflog can be used to identify and restore lost or deleted commits, equipping us with the ability to recover from potential setbacks in their projects.

03

Troubleshooting Scenarios

We can learn to use the reflog to troubleshoot and recover from accidental changes, enhancing their resilience in software development.

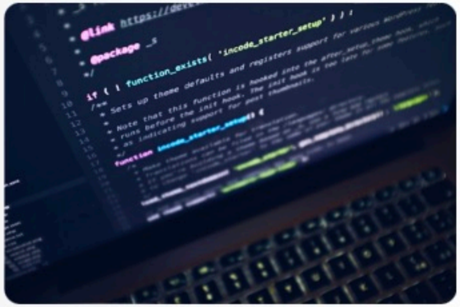
Advanced Git Diff Techniques

Comparing Different Commits

Git diff can be used to compare changes between different commits, branches, or tags, enabling us to understand the evolution of our project and track modifications effectively.

Visualizing Changes

Utilize graphical tools to visualize and understand differences between commits and branches, providing students with a comprehensive view of their project history.



Enhancing Code Understanding

We can utilize these techniques to gain insights into our code changes and collaborate effectively with our peers.

Section 3

Practical Applications of Git

Real-world Examples and Demonstrations

01

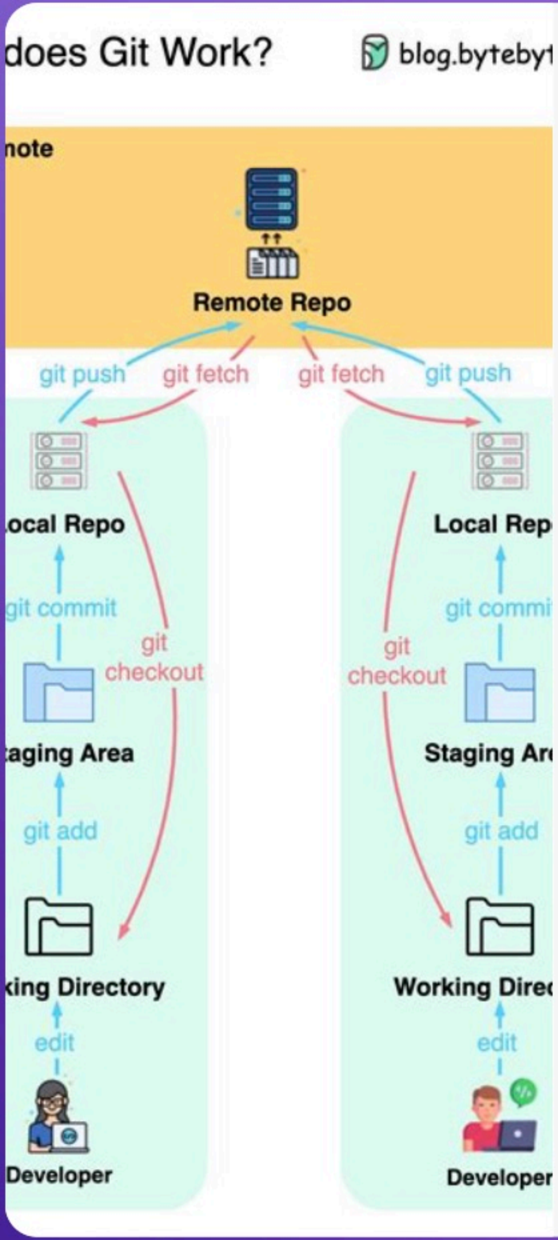
Example 1: Collaborative Workflow

Showcase how Git enables collaborative development through branching, merging, and conflict resolution, illustrating the practical relevance of version control in team projects.

02

Example 2: Feature Development

Demonstrate the use of Git for feature branching, iterative development, and code review processes, providing us with real-world insights into project management with Git.



Live Demo: Git Workflow in Action

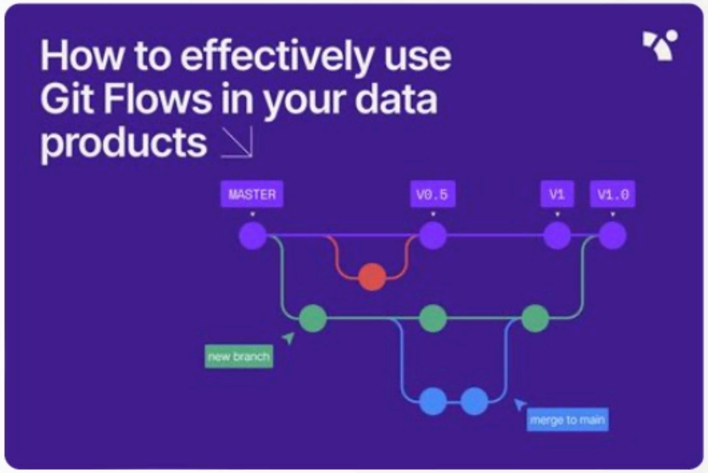
Key Stages of Git Workflow

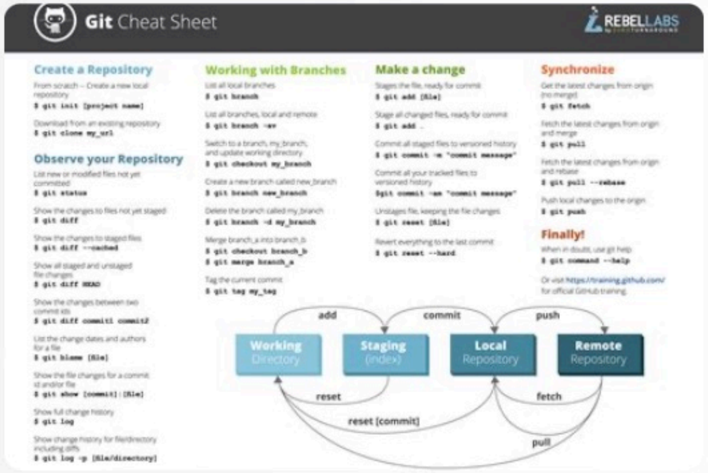
Git workflow encompasses a series of stages, from code creation to deployment, enabling seamless collaboration and version control. Key stages include: Creating branches, Making changes, Staging changes, Committing changes, Merging branches, Pushing changes to a remote repository.

Conclusion

Through this live demonstration, we've witnessed firsthand how Git workflow empowers teams to efficiently manage code changes and collaborate effectively. Understanding and implementing Git workflow principles can greatly enhance productivity and code quality in software development projects.

Visual Aids and Code Snippets





Git Cheat Sheet

REBELLABS

Create a Repository

From scratch -- Create a new local repository

```
$ git init [project name]
```

Download from an existing repository

```
$ git clone my_url
```

Observe your Repository

List new or modified files not yet committed

```
$ git status
```

Show the changes to files not yet staged

```
$ git diff
```

Show the changes to staged files

```
$ git diff --cached
```

Show all staged and unstaged file changes

```
$ git diff HEAD
```

Show the changes between two commit ids

```
$ git diff commit1 commit2
```

List the change dates and authors for a file

```
$ git blame [file]
```

Show the file changes for a commit id and/or file

```
$ git show [commit]:[file]
```

Show full change history

```
$ git log
```

Show change history for file/directory including diffs

```
$ git log -p [file/directory]
```

Working with Branches

List all local branches

```
$ git branch
```

List all branches, local and remote

```
$ git branch -av
```

Switch to a branch, my_branch, and update working directory

```
$ git checkout my_branch
```

Create a new branch called new_branch

```
$ git branch new_branch
```

Delete the branch called my_branch

```
$ git branch -d my_branch
```

Merge branch_a into branch_b

```
$ git checkout branch_b
$ git merge branch_a
```

Tag the current commit

```
$ git tag my_tag
```

Make a change

Stages the file, ready for commit

```
$ git add [file]
```

Stage all changed files, ready for commit

```
$ git add .
```

Commit all staged files to versioned history

```
$ git commit -m "commit message"
```

Commit all your tracked files to versioned history

```
$git commit -am "commit message"
```

Unstages file, keeping the file changes

```
$ git reset [file]
```

Revert everything to the last commit

```
$ git reset --hard
```

Synchronize

Get the latest changes from origin (no merge)

```
$ git fetch
```

Fetch the latest changes from origin and merge

```
$ git pull
```

Fetch the latest changes from origin and rebase

```
$ git pull --rebase
```

Push local changes to the origin

```
$ git push
```

Finally!

When in doubt, use git help

```
$ git command --help
```

Or visit https://training.github.com/ for official GitHub training.

Visual Representation

This image visually depicts fundamental Git processes: branching, merging, and committing. Branching allows isolated work on features or fixes, while merging integrates changes into the main codebase. Each commit captures specific modifications, fostering collaboration and project transparency. Together, these Git workflow components streamline development, ensuring efficient code management and teamwork.

Git Commands

The Git cheat sheet provides quick-reference commands for fundamental Git operations, from initializing repositories to branching, merging, and more. Use it to streamline your Git workflow and enhance productivity at every stage of your development process.

Conclusion

Summarizing Key Takeaways

In summary, Git's robust capabilities empower developers to collaborate seamlessly, iterate efficiently, and maintain code integrity. Embracing Git's principles fuels innovation and accelerates project delivery. Thank you for joining us on this journey; let's continue to explore and leverage Git's full potential together.

Thank You

Prepared By- Mauryavardhan Singh (23ME10048)