Git Basics

1. Introduction to Git

Git is a distributed version control system that helps developers track changes in source code and collaborate on projects.

---

2. Basic Git Configuration

```
git config --global user.name "Your Name"
git config --global user.email "you@example.com"
```

These commands set your global identity used in all repositories.

---

3. Creating a Git Repository

```
mkdir GitLearning
cd GitLearning
git init
```

This creates a new local Git repository.

---

4. Git Status and Adding Files

```
touch hello.txt
git status          # See current changes
git add hello.txt   # Stage a file
```

---

5. Committing Changes

```
git commit -m "Initial commit"
```

This records staged changes to the repo.

---

6. Making Changes and New Commits

```
echo "I am shreya." >> hello.txt
git add hello.txt
git commit -m "Modified hello.txt"
```

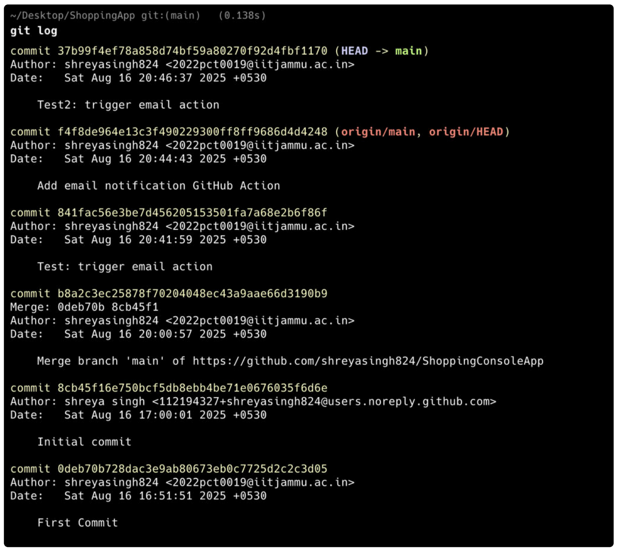
You can continue to edit files and make new commits as needed.

---

7. Git Log and History

```
git log
```

Shows the commit history.



```
~/Desktop/ShoppingApp git:(main)  (0.138s)
git log
commit 37b99f4ef78a858d74bf59a80270f92d4fbf1170 (HEAD -> main)
Author: shreyasingh824 <2022pct0019@iitjammu.ac.in>
Date:   Sat Aug 16 20:46:37 2025 +0530

    Test2: trigger email action

commit f4f8de964e13c3f490229300ff8ff9686d4d4248 (origin/main, origin/HEAD)
Author: shreyasingh824 <2022pct0019@iitjammu.ac.in>
Date:   Sat Aug 16 20:44:43 2025 +0530

    Add email notification GitHub Action

commit 841fac56e3be7d456205153501fa7a68e2b6f86f
Author: shreyasingh824 <2022pct0019@iitjammu.ac.in>
Date:   Sat Aug 16 20:41:59 2025 +0530

    Test: trigger email action

commit b8a2c3ec25878f70204048ec43a9aae66d3190b9
Merge: 0deb70b 8cb45f1
Author: shreyasingh824 <2022pct0019@iitjammu.ac.in>
Date:   Sat Aug 16 20:00:57 2025 +0530

    Merge branch 'main' of https://github.com/shreyasingh824/ShoppingConsoleApp

commit 8cb45f16e750bcf5db8ebb4be71e0676035f6d6e
Author: shreya singh <112194327+shreyasingh824@users.noreply.github.com>
Date:   Sat Aug 16 17:00:01 2025 +0530

    Initial commit

commit 0deb70b728dac3e9ab80673eb0c7725d2c2c3d05
Author: shreyasingh824 <2022pct0019@iitjammu.ac.in>
Date:   Sat Aug 16 16:51:51 2025 +0530

    First Commit
```

---

8. Git Diff

```
git diff                # See unstaged changes
git diff --staged       # See staged but uncommitted changes
```

Use `git diff` to view what exactly changed between commits or working tree.

---

9. What is a Remote?

A **remote** is a version of your repository hosted on the internet or network. It allows you to collaborate with others.

```
git remote add origin https://github.com/username/repo.git
git remote -v           # List remotes
```

---

10. Git Push (Uploading to Remote)

```
git push -u origin master
```

Pushes your local commits to the remote repository.

---

11. Git Pull (Fetching and Merging Remote Changes)

```
git pull
```

Fetches and integrates remote changes into your local branch.

---

12. Git Clone (Copying a Remote Repo)

```
git clone https://github.com/username/repo.git
```

Creates a local copy of the remote repository.

---

13. Git Fork (GitHub Feature)

Forking is a GitHub operation to create a personal copy of someone else's repository. It enables experimentation without affecting the original project.

---

14. Workflow Tip: Pull Often

Frequently use:

```
git pull
```

This ensures you're always working with the latest code and reduces chances of merge conflicts.

---

15. Git Ignore

To prevent certain files from being tracked:

1. Create `.gitignore`
2. Add entries like:

```
*.log
*.class
.idea/
```

---

16. Git Stash

Temporarily save uncommitted changes:

```
git stash              # Save changes
# Do something else
git stash pop          # Reapply changes
```

---

17. Git Rebase

```
git rebase branchname
```

Re-applies your changes on top of another branch. Used to create cleaner history.

---

18. Git Squash (with Rebase)

```
git rebase -i HEAD~3
```

Squashes multiple commits into one. Use during cleanup before merging.

---

19. Reverting Commits

```
git revert <commit-id>
```

Used to undo a commit by creating a new commit that reverses it.

---

20. Resetting to Previous Commits

```
git reset --hard <commit-id>
```

Rolls back to a specific commit (use with caution).

---

21. Working with Branches

```
git branch first_branch     # Create a branch
git checkout first_branch   # Switch to branch
git add .
git commit -m "Changes in first_branch"
```

---

22. Merging Branches

```
git checkout master
git merge first_branch
```

This combines changes from one branch into another (e.g., feature branch into main branch).

---

23. Merge Conflicts (with Example)

Scenario:

1. `master` has line "I am shreya."
2. `first_branch` changes it to "I am shreya from branch."
3. When merging, Git detects a conflict in `hello.txt`.

Git will show this:

```
<<<<<<< HEAD
I am shreya.
=======
I am shreya from branch.
>>>>>>> first_branch
```

You must manually edit the file, then run:

```
git add hello.txt
git commit -m "Resolved merge conflict"
```

---

24. Finding Files

```
find . -name "hello.txt"
```

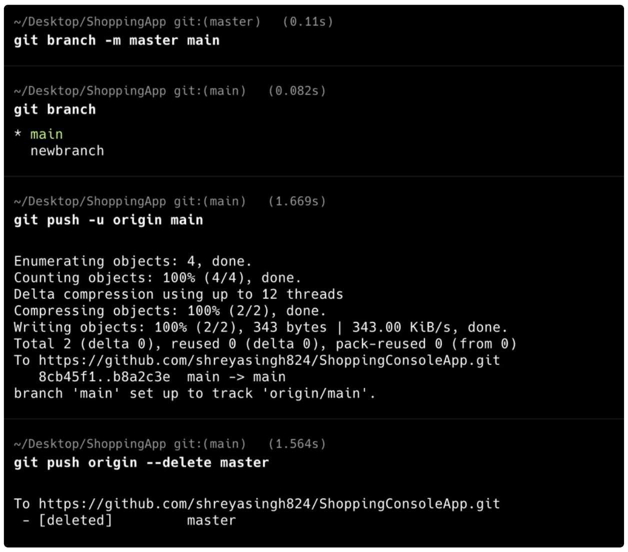
This finds `hello.txt` even inside hidden folders.

Rename master to main

git branch -m master main

git push -u origin main

git push origin --delete master



```
~/Desktop/ShoppingApp git:(master)   (0.11s)
git branch -m master main

~/Desktop/ShoppingApp git:(main)   (0.082s)
git branch
* main
  newbranch

~/Desktop/ShoppingApp git:(main)   (1.669s)
git push -u origin main

Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 12 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 343 bytes | 343.00 KiB/s, done.
Total 2 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/shreyasingh824/ShoppingConsoleApp.git
   8cb45f1..b8a2c3e  main -> main
branch 'main' set up to track 'origin/main'.

~/Desktop/ShoppingApp git:(main)   (1.564s)
git push origin --delete master

To https://github.com/shreyasingh824/ShoppingConsoleApp.git
 - [deleted]         master
```

25. Useful Git Commands Summary

Purpose	**Command**
Initialize Repo	`git init`
Stage Files	`git add filename`
Commit Changes	`git commit -m "message"`
View Status	`git status`
View History	`git log`
View Diffs	`git diff`
Create Branch	`git branch branchname`
Switch Branch	`git checkout branchname`
Merge Branch	`git merge branchname`
Revert Commit	`git revert <commit-id>`
Hard Reset	`git reset --hard <commit-id>`

Add Remote	`git remote add origin <repo-url>`
Push to Remote	`git push -u origin branchname`
Pull from Remote	`git pull`
Clone Repo	`git clone <repo-url>`
Stash Changes	`git stash` / `git stash pop`
Rebase Branch	`git rebase branchname`
Squash Commits	`git rebase -i HEAD~n`

---

26. GitHub Actions and Automation (CI/CD)

GitHub Actions: Complete Guide to Automation and CI/CD Pipelines

1. What are GitHub Actions?

GitHub Actions is a **workflow automation** feature built into GitHub that lets you:

- Run **CI/CD pipelines** (Continuous Integration / Continuous Deployment)
- Automate **builds, tests, notifications, and deployments**
- Respond to GitHub events (push, pull request, issue creation, etc.)

Workflows are defined using YAML in `.github/workflows/`.

---

2. Basic Structure of a GitHub Action Workflow

```
name: <Workflow Name>           # Descriptive name for your workflow

on:                             # Trigger conditions
  push:
  pull_request:
```

```
jobs:                              # One or more jobs to run
  job_name:
    runs-on: ubuntu-latest     # Type of machine to run on
    steps:
      - name: Step name        # Human-friendly name of step
        uses: action-name@v1   # Use an action from 
marketplace
        run: some command      # OR directly run a command
```

This structure allows defining what event starts the workflow, which jobs run, and in what order steps execute within jobs.

---

3. Triggers (Events)

You can trigger workflows based on:

- `push`
- `pull_request`
- `workflow_dispatch` (manual button)
- `schedule` (cron jobs)
- `release`, `issue_comment`, etc.

Example:

```
on:
  push:
    branches:
      - main
```

---

4. Jobs and Steps

Each workflow contains **jobs**, and each job contains **steps**.

- Jobs run on their own virtual machine
- Steps run in sequence inside that job

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Hello World"
```

---

5. Complete Email Notification Workflow (Original Project)

This example sends an email on every **push**, **pull request**, or **new commit**:

```
name: Notify on Push or PR

on:
  push:
  pull_request:

jobs:
  notify:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout repository
        uses: actions/checkout@v3

      - name: Send email notification
        uses: dawidd6/action-send-mail@v3
        with:
          server_address: smtp.gmail.com
          server_port: 465
          username: ${{ secrets.EMAIL_USERNAME }}
          password: ${{ secrets.EMAIL_PASSWORD }}
          subject: GitHub Repo Update
          to: 2022pct0019@iitjammu.ac.in
          from: GitHub Actions shreyarathour824@gmail.com
          body: |
            🚀 GitHub event detected!

            Repository: ${{ github.repository }}
            Branch: ${{ github.ref }}
            Event: ${{ github.event_name }}
            Commit: ${{ github.sha }}
            Actor: ${{ github.actor }}

            View commit: https://github.com/${{
github.repository }}/commit/${{ github.sha }}

```

---

6. How to Generate App Password (for Gmail)

Gmail no longer allows access from "less secure apps." You must use an **App Password**, which is a special 16-character password:

Steps:

1. Enable **2-Step Verification** on your Google Account: https://myaccount.google.com/security
2. Go to: https://myaccount.google.com/apppasswords
3. Select **Mail** as the app, and **Other (Custom name)** as the device.
4. Generate the password.
5. Copy the **16-character password** (with spaces) — use it **as-is**.

---

7. Add GitHub Secrets

Go to your repo:

1. **Settings** → **Secrets and Variables** → **Actions**
2. Click **New repository secret**
3. Add:
   - `EMAIL_USERNAME` → your email (e.g. shreyarathour824@gmail.com)

- `EMAIL_PASSWORD` → the 16-character Gmail App Password

---

8. How to Push the YAML File

From terminal:

```
mkdir -p .github/workflows
nano .github/workflows/email-on-update.yml   # or use any editor
# Paste the workflow YAML content here

git add .github/workflows/email-on-update.yml
git commit -m "Add GitHub Action for email notification"
git push
```

---

9. Test the Workflow

1. Make any commit:

   ```
   echo "test" >> test.txt
   git add test.txt
   git commit -m "Test commit"
   git push
   ```

2. Go to GitHub repo → **Actions** tab → Check for green ✅ run.
3. Check your email inbox or spam folder.

---

10. Common Errors

Error Type	Cause	Solution
Authentication	Wrong email or password	Use App Password (not Gmail login password)
No workflow run	File name or path wrong	Ensure it's `.github/workflows/file.yml`

Email not received	Spam or wrong receiver	Check spam or correct `to:` field

---

11. How to Change Functionality

To change what the GitHub Action does:

✅ **Change the trigger (when it runs):**

```
on:
  push:
    branches:
      - main
```

→ Only triggers on pushes to `main`

✅ **Change the recipient email:**

```
to: new-recipient@example.com
```

✅ **Change what the email says:**

```
subject: "New Deployment Notification"
body: |
  🚨 New code was pushed by ${{ github.actor }}!
```

✅ **Add conditions (e.g., run only if file changes):**

```
on:
  push:
    paths:
      - "src/**"
```

→ Only run if files in `src/` change

✅ **Add new jobs (e.g., build, test):**

```
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - run: echo "Running tests..."

  notify:
    needs: test
    runs-on: ubuntu-latest
    steps:
      - run: echo "Sending email..."
```

→ Runs test first, then sends email

Auto-Merge Pull Requests Workflow

This GitHub Action **automatically merges pull requests** if all required checks pass and conditions are met.

```
name: Auto Merge PRs

on:
  pull_request:
    types:
      - labeled
      - opened
      - synchronize

permissions:
  pull-requests: write
  contents: write

jobs:
  automerge:
    runs-on: ubuntu-latest
    if: github.event.pull_request.user.login != 'github-actions[bot]'
    steps:
      - name: Auto-merge PR
        uses: pascalgn/automerge-action@v0.15.6
        env:
          GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
```

💡 **How It Works:**

- It runs when a pull request is **opened**, **labeled**, or **synchronized** (updated).
- It checks that the PR is not created by a bot.
- It uses `pascalgn/automerge-action` to automatically merge once all status checks pass.

🛠️ **Requirements:**

- The repo must have branch protection rules configured.
- You may label PRs with something like `automerge` to trigger the behavior.