Git & GitHub

🧠 Git কী?

Git হলো একটি **Version Control System (VCS)** — মানে এমন একটা টুল, যা কোডের পরিবর্তন (changes) গুলো ট্র্যাক করে রাখে।

➞ তুমি যখন কোনো প্রজেক্টে কাজ করো, তখন হয়তো বারবার কোডে পরিবর্তন আনতে হয়।
➞ Git সেই পরিবর্তনগুলোর রেকর্ড রাখে — কে, কখন, কোন লাইনে কী পরিবর্তন করেছে।
➞ ফলে তুমি চাইলে পুরোনো কোডে ফিরে যেতে পারো, অন্যদের সঙ্গে একসাথে কাজ করতে পারো, এমনকি কোড merge করতেও পারো।

🔹 Git দিয়ে কী করা যায়:

1. **Code version সংরক্ষণ করা** — আগের version গুলো দেখতে পারো।
2. **Teamwork সহজ করা** — একাধিক ডেভেলপার একসাথে একই প্রজেক্টে কাজ করতে পারে।
3. **Backup রাখা** — কোড হারানোর ভয় থাকে না।
4. **Branch তৈরি করা** — main কোড নষ্ট না করে নতুন feature develop করা যায়।

🧩 Git এর কিছু গুরুত্বপূর্ণ কমান্ড:

Command	কাজ
git init	নতুন git repository শুরু করে

git status	কোন ফাইল পরিবর্তিত হয়েছে সেটা দেখায়
git add file_name	পরিবর্তন stage করে
git commit -m "message"	পরিবর্তন save করে commit হিসেবে
git log	সব commit history দেখায়
git branch	সব branch দেখায়
git checkout branch_name	অন্য branch এ switch করে
git merge branch_name	অন্য branch এর কোড main এ মেশায়

🧭 GitHub কী?

GitHub হলো একটা **online platform (website)**, যেখানে তুমি তোমার **Git project গুলো সংরক্ষণ ও শেয়ার** করতে পারো।

➡ সহজভাবে বললে —
Git হলো **লোকাল version control system** (তোমার কম্পিউটারে চলে)।
আর **GitHub** হলো **ক্লাউড বা অনলাইন প্ল্যাটফর্ম**, যেখানে তুমি সেই কোড **upload করে অনলাইনে রাখো**।

🔹 GitHub দিয়ে কী করা যায়:

1. **Code অনলাইনে সংরক্ষণ করা** ☁️
2. **Team members এর সঙ্গে কাজ করা** 👨‍💻👩‍💻
3. **Open source project contribute করা** 🌍
4. **Issue tracking, discussion, pull request করা** 📜
5. **Portfolio হিসেবে ব্যবহার করা** – চাকরির জন্য অনেক গুরুত্বপূর্ণ 💼

🔗 Git ↔ GitHub সম্পর্ক

- Git = তোমার লোকাল টুল
- GitHub = তোমার কোডের অনলাইন হোম

তুমি Git দিয়ে কাজ করবে, আর GitHub এ সেই কাজ **push** করে রাখবে, যাতে অন্যরা দেখতে বা কাজ করতে পারে।

GitHub-এ টিম প্রজেক্টে সহযোগিতা করবেন, ব্রাঞ্চ তৈরি করবেন, কোড পুশ করবেন, এবং অন্যদের সাথে কিভাবে কাজ সামলাবেন

1) প্রথম সেটআপ — পরিচিতি ও প্রয়োজনীয়তা

- **Git ইনস্টল করুন** (Windows/Mac/Linux). নিশ্চিত করুন git --version কাজ করে।
- GitHub-এ **অ্যাকাউন্ট** থাকবে। টিম হলে সাধারণত একটি Organization তৈরি করা হয় (optional) — এতে repo, teams, permissions সহজে কন্ট্রোল করা যায়।
- লোকাল মেশিনে SSH **বা** HTTPS সেটআপ:
  - SSH সুবিধা: একবার সেটআপ করলে বারবার পাসওয়ার্ড লাগবে না।
  - HTTPS: সহজ, কিন্তু প্রতি push/pull-এ credentials লাগতে পারে (token ব্যবহার আশা করুন)।
- Git global config:

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

2) GitHub-এ রেপোজিটরি তৈরি ও টিম কনফিগারেশন (UI)

1. GitHub → New repository → নাম, বর্ণনা দিন (public/private)।
2. Organization ব্যবহার করলে: Settings → Manage access → **Invite teams/users** (Collaborators)।
3. **Teams** তৈরি করলে (Organization → Teams) আপনি নির্দিষ্ট অনুমতি (Read, Write, Admin) সেট করতে পারেন।
4. main বা master-কে **protected branch** বানাতে চাইলে: Repo → Settings → Branches → Add rule → main → Require pull request reviews, require status checks ইত্যাদি চালু করুন। (এভাবে কেউ সরাসরি ধাক্কা (push) দেবে না।)

3) লোকালি রেপো ক্লোন করা

```bash
# SSH
git clone git@github.com:org-or-user/project-x.git

# অথবা HTTPS
git clone https://github.com/org-or-user/project-x.git
```

- ক্লোন করার পর cd project-x এবং git remote -v দেখে remote ঠিক আছে কিনা চেক করুন।

4) ব্রাঞ্চিং কনভেনশন ও কেমন নাম দেবেন (Best practices)

- সাধারণ নামকরণ:
  - ফিচার: feature/login, feature/payment-int
  - বাগ ফিক্স: bugfix/typo-on-home
  - হটফিক্স: hotfix/urgent-fix
  - রিলিজ: release/v1.2.0

- ছোট, স্পষ্ট নাম রাখবেন; কাজ সংক্রান্ত টিকিট নম্বর থাকলে যোগ করলে ভালো — feature/JIRA-123-add-login.

---

5) নতুন ব্রাঞ্চ তৈরি করে কাজ করা (স্টেপ-বাই-স্টেপ)

1. লেটেস্ট কোড নিন:

```
git checkout main
git pull origin main
```

2. নতুন ব্রাঞ্চ বানান:

```
bash

git checkout -b feature/login
```

-b মানে create + switch।

3. কোড বদলান — ফাইল যুক্ত/পরিবর্তন/ডিলিট।
4. পরিবর্তনগুলো স্টেজ করুন:

```
bash

git status
git add file1 file2
# অথবা সবগুলো
git add .
```

5. সুন্দর commit message দিন:

```
bash

git commit -m "feature(login): add login form and basic validation"
```

commit message guideline: প্রথম অংশ সংক্ষিপ্ত (50 chars), প্রয়োজনে বডি লিখুন git commit

এ editor খুলে। ব্যবহার: Conventional Commits (optional) — feat:, fix:, chore: ইত্যাদি।

6. লোকালি থেকেও ব্রাঞ্চ পুশ করুন (প্রথমবার remote এ branch না থাকলে):

```
bash

git push -u origin feature/login
```

-u দিয়ে upstream সেট করে দেয়, ভবিষ্যতে শুধু git push যাবে।

6) GitHub এ Pull Request (PR) করা — কোড রিভিউ

1. GitHub repo → Pull requests → New pull request → select base (main) এবং compare (feature/login)।
2. PR শিরোনাম ও বর্ণনায় টাস্ক ব্যাখ্যা করুন: কাজ কি, কি পরিবর্তন এসেছে, কিভাবে টেস্ট করবেন।
3. Reviewers এবং Assignees নিয়োগ করুন; প্রয়োজনীয় টীম/লোক add করুন।
4. যদি CI (GitHub Actions) থাকে, সেটি চলে কিনা দেখুন—সব test/status pass হওয়া উচিত।
5. রিভিউয়াররা কমেন্ট করলে প্রয়োজনীয় পরিবর্তন করে আবার push করুন — PR আপডেট হয়ে যাবে।

7) রিভিউ প্রক্রিয়া, মন্তব্য ও পরিবর্তন

- রিভিউয়াররা কোড লাইনে মন্তব্য করতে পারে — আপনি ওই লাইনে পরিবর্তন করে commit করলে PR-এ আপডেট হবে।
- ছোট পরিবর্তন হলে squash and merge করতে পারেন (কোনো ছোট ফিচারের ক্ষেত্রে)। বড় হলে merge বা rebase and merge সিদ্ধান্ত টীম নেবে।

- **Conflict** এলে লোকালে resolve করতে হবে (নিচে conflict সমাধান দেখানো আছে)।

8) Merge **করার আগে লোকালি আপডেট রাখা** (recommended)

মার্জ করার সময় conflict কমানোর জন্য সাধারণ ধারা:

```bash
# আপনার ফিচার ব্রাঞ্চে থাকলে
git checkout feature/login
# রিমোট main থেকে নিন
git fetch origin
git rebase origin/main
# অথবা merge করলে
# git merge origin/main

# conflict হলে ফাইল সমাধান করে
git add <resolved-files>
git rebase --continue   # যদি rebase করছিলেন

# এরপর push (force required after rebase)
git push --force-with-lease origin feature/login
```

rebase করলে commit history পরিষ্কার থাকে; তবে shared branch হলে সাবধানে ব্যবহার করুন।

9) Conflict **কি, কিভাবে সমাধান করবেন**

- Conflict ঘটে যখন একই ফাইলে একই লাইনে দুইজন ভিন্ন পরিবর্তন করেছে। Git আপনাকে conflict-markers দেখাবে: `<<<<<<<`, `=======`, `>>>>>>>`
- Example:

```
text

<<<<<<< HEAD
your local change
=======
incoming change from main
>>>>>>> origin/main
```

আপনি সিদ্ধান্ত নেবেন কোনটা রাখতে হবে বা দুটো মিলিয়ে নতুন কোড লিখবেন → ফাইল সেভ করবেন → git add .

→ git commit -m"(বা git rebase --continue)"

→ git push

10) Pull vs Fetch vs Pull --rebase (সংক্ষেপে)

- git fetch: রিমোট-এর লেটেস্ট ডাটা নামায় কিন্তু merge করে না। (নিরাপদ)
- git pull: fetch + merge (ডিফল্টভাবে)।
- git pull --rebase: fetch + rebase (cleaner history)।
- টিমে ঐক্যমত থাকলে pull --rebase ব্যবহার করলে history linear থাকে।

---

11) Code review best practices (টিপস)

- ছোট PR রাখুন (৫০–২০০ লাইন পরিবর্তনের মধ্যে যদি সম্ভব) — রিভিউ দ্রুত হয়।
- PR টাইটল স্পষ্ট। PR বর্ণনায় test steps লিখুন।
- নিজে PR করা আগে নিজে কোড পড়ুন ও লাকাল টেস্ট চালান।
- Reviewers কে ট্যাগ করুন; লেবেল ব্যবহার করুন (ready for review, WIP)।
- Automated tests, linters (ESLint, Prettier), এবং CI চালুন — PR merge এর আগে passing প্রয়োজন।

---

12) Protected branches, required reviews, status checks

- Repo Settings → Branches → Branch protection rule:
  - Require pull request reviews before merging (n reviewers)
  - Require status checks to pass (CI builds, tests)
  - Require signed commits (optional)
  - Restrict who can push to branch (prevent direct pushes)

---

13) Fork workflow (open-source বা external contributors)

- Contributor fork করে কাজ করে (their own repo), branch তৈরি, push করে → Upstream repo-তে PR দেয়।
- Maintainer review ও merge করে।

---

14) Common useful git commands (quick cheat-sheet)

```
# status, diff
git status
git diff              # unstaged changes
git diff --staged     # staged changes

# create/switch branch
git checkout -b feature/x
git checkout main
```

```
# update main
git pull origin main

# add, commit, push
git add .
git commit -m "msg"
git push origin feature/x

# fetch remote updates
git fetch origin

# rebase vs merge
git rebase origin/main
git merge origin/main

# undo last local commit (soft)
git reset --soft HEAD~1

# stash changes temporarily
git stash
git stash pop

# show log
```

```
git log --oneline --graph --decorate --all


# tag
git tag -a v1.0 -m "release 1.0"
git push origin v1.0
```

15) PR merge strategies ও তাদের প্রভাব

- **Merge commit**: সকল commit history ধরে রেখে merge commit তৈরি করে। (history complete কিন্তু noisy)
- **Squash and merge**: সব commit একটায় squash হয়ে আসে — history পরিষ্কার কিন্তু individual commits মুছে যায়।
- **Rebase and merge**: আপনার commits rebase হয়ে main-এ linear ভাবে যোগ হয় — পরিষ্কার কিন্তু ইতিহাস বদলায় না।

টীম কবে কোনটা ব্যবহার করবে সেটা পলিসি হিসেবে ঠিক করুন।

---

16) Continuous Integration (CI) ও automation (সংক্ষেপে)

- GitHub Actions/Travis/CircleCI ব্যবহার করে PR এ automated tests চালান।
- Lint, unit tests, build tests, security scans চালিয়ে passing হলে merge করা ভাল।
- Auto-merge on green (option) থাকতে পারে — সাবধানে enable করুন।

---

17) Issues, Project boards ও টাস্ক ট্র্যাকিং

- **Issues**: টাস্ক/বাগ আলাদা করে রাখে। PR-এ Issue link করতে পারেন (Fixes #12) — merge হলে auto-close।

- **Projects (Kanban)**: To-do / In progress / Done কলাম সহ টাস্ক organize করতে পারবেন।
- PRs, Releases, Milestones ব্যবহার করে roadmap ট্র্যাক করুন।

---

18) সিকিউরিটি ও পারমিশন বিষয়

- Private repo হলে প্রত্যেককে Invite করতে হবে।
- টিম-লেভেল পারমিশন: Read / Triage / Write / Maintain / Admin।
- Sensitive data কখনো রিপোতে commit করবেন না (API keys, passwords)। যদি হয়ে যায় → সেটা invalidate করে git filter-repo বা BFG দিয়ে ইতিহাস থেকে সরান।

---

19) সাধারণ সমস্যাগুলি ও সমাধান

- **Push rejected (protected branch)**: PR তৈরি করুন, সরাসরি push না করুন।
- **Authentication failed**: SSH key যোগ করুন বা Personal Access Token (PAT) ব্যবহার করুন (HTTPS)।
- **Merge conflicts**: লোকাল conflict resolve করে push করুন।
- **Forgot to pull before push**: git pull --rebase পরে git push (বা merge)।

---

20) উদাহরণ End-to-End (তুলনামূলক দ্রুত গল্প)

1. main-কে আপডেট করে নিন: git checkout main && git pull origin main
2. git checkout -b feature/navbar
3. কোড লিখুন, git add . && git commit -m "feat(navbar): responsive navbar"
4. git push -u origin feature/navbar

5. GitHub-এ গিয়ে PR খুলুন → reviewers add করুন → CI পাস করুন → feedback এডজাস্ট করুন → আবার push করুন।
6. সব ঠিক থাকলে Merge করুন (Squash/Merge বা টীমের পলিসি অনুযায়ী)।
7. লোকালি main পেতে: git checkout main && git pull origin main → তারপর git branch -d feature/navbar (delete local) এবং git push origin --delete feature/navbar (delete remote) (optional)।