গিটিং হ্যান্ডনোট

আকন এম হাসিব

অধ্যায় ১: ভূমিকা

1.1 আমরা কেন ভার্সন কন্ট্রোলের প্রয়োজন অনুভব করি?

ধরুন আপনি একটি প্রজেক্টে কাজ করছেন—হয়তো একটি লেখা, অথবা প্রোগ্রামিং প্রজেক্ট। প্রথম দিন কাজ শুরু করে একটি ফাইল সেভ করলেন project.docx নামে। পরের দিন কিছু পরিবর্তন করলেন, কিন্তু আগের ভার্সন রাখতে চাইলেন, তাই নতুন নামে সেভ করলেন project_v2.docx। আবার কয়েকদিন পর আরও কিছু পরিবর্তন করলেন, এবার নাম দিলেন project_final.docx।

কিছুদিন পর ফোল্ডারে দেখা গেল এরকম অনেকগুলো ফাইল:

- project_final2.docx
- project_final_revised.docx
- project_final_revised_new.docx

এখন আপনি নিজেই বুঝতে পারছেন না কোন ফাইলটি আসল, কোথায় কী পরিবর্তন করেছেন। আর যদি একই প্রজেক্টে তিনজন মিলে কাজ করেন, তবে অবস্থা আরও জটিল হয়ে যাবে।

☞ **এ সমস্যার সমাধানই হলো ভার্সন কন্ট্রোল সিস্টেম (Version Control System)।**

1.2 ভার্সন কন্ট্রোল কী?

ভার্সন কন্ট্রোল (Version Control) হলো এমন একটি ব্যবস্থা, যা আপনার ফাইলের প্রতিটি পরিবর্তন ট্র্যাক করে রাখে।

- কে পরিবর্তন করেছে,
- কখন করেছে,
- কী পরিবর্তন করেছে—

সবকিছুই এতে সংরক্ষিত থাকে। প্রয়োজনে আপনি আগের কোনো ভার্সনে ফিরে যেতে পারবেন, অথবা দেখতে পারবেন কোন লাইনে কী যোগ বা বাদ হয়েছে।

1.3 Git কী?

Git হলো একটি জনপ্রিয় ভার্সন কন্ট্রোল সফটওয়্যার, যা ২০০৫ সালে **Linus Torvalds** (Linux এর স্রষ্টা) তৈরি করেন। এটি ওপেন সোর্স এবং আজ বিশ্বের প্রায় সব সফটওয়্যার কোম্পানি ব্যবহার করছে।

Git কে বলা হয় **distributed version control system** কারণ এতে প্রত্যেকের কাছে পুরো প্রজেক্টের ইতিহাসের কপি থাকে। অর্থাৎ, আপনার কম্পিউটারে ইন্টারনেট ছাড়াই Git ব্যবহার করতে পারবেন।

1.4 Git কেন দরকার?

Git ব্যবহার করলে অনেক সুবিধা পাওয়া যায়, যেমন:

- ✅ **ইতিহাস সংরক্ষণ**: কোন ফাইল কবে কীভাবে বদলেছে, তার রেকর্ড থাকে।
- ✅ **সহযোগিতা (Collaboration)**: একসাথে অনেকজন একই প্রজেক্টে কাজ করতে পারে।
- ✅ **ব্যাকআপ সুবিধা**: যেহেতু প্রত্যেকের কাছে পুরো রিপোজিটরির কপি থাকে, তাই ডেটা হারানোর সম্ভাবনা কম।

- ✅ **ফ্লেক্সিবিলিটি**: নতুন ফিচার টেস্ট করার জন্য আলাদা branch ব্যবহার করা যায়।

1.5 Git বনাম GitHub

অনেকে Git আর GitHub কে একই জিনিস ভাবে। কিন্তু আসলে এগুলো আলাদা:

- **Git**: সফটওয়্যার টুল, যা আপনার কম্পিউটারে থাকে এবং ফাইল/প্রজেক্টের ইতিহাস ট্র্যাক করে।
- **GitHub**: একটি অনলাইন প্ল্যাটফর্ম যেখানে Git দিয়ে তৈরি রিপোজিটরি (repository) রাখা যায়।

সহজ উদাহরণ:

- Git হলো আপনার **কম্পিউটার** (যেখানে ফাইল রাখেন)।
- GitHub হলো **Google Drive** বা **Dropbox** এর মতো একটি জায়গা (যেখানে ফাইল অনলাইনে শেয়ার করতে পারেন)।

1.6 বাস্তব জীবনের উদাহরণ

- **ছাত্রছাত্রীদের জন্য**: অ্যাসাইনমেন্টের বিভিন্ন সংস্করণ ম্যানেজ করা।
- **প্রোগ্রামারদের জন্য**: বড় সফটওয়্যার প্রজেক্টে একসাথে টিম হিসেবে কাজ করা।
- **অফিসের কাজে**: রিপোর্ট বা ডকুমেন্টে কে কী পরিবর্তন করেছে, সেটা ট্র্যাক করা।
- **ওপেন সোর্স প্রজেক্টে**: সারা বিশ্বের মানুষ একসাথে কাজ করতে পারে GitHub এর মাধ্যমে।

1.7 এই বই থেকে কী শিখবেন

এই বইয়ে আমরা ধাপে ধাপে শিখবো:

1. **Git এর বেসিক** – কীভাবে add, commit, log ইত্যাদি ব্যবহার করতে হয়।
2. **GitHub এর বেসিক** – অ্যাকাউন্ট খোলা, রিপোজিটরি তৈরি, ফাইল আপলোড করা।
3. **টিমওয়ার্ক** – একাধিক ডেভেলপার মিলে কাজ করা, Pull Request, Merge করা।
4. **অ্যাডভান্সড Git** – rebase, stash, cherry-pick এর মতো টেকনিক।
5. **GitHub এর বিশেষ ফিচার** – Actions (automation), Pages (ওয়েবসাইট), Projects (টাস্ক ম্যানেজমেন্ট)।

1.8 অনুশীলনী

1. আপনার নিজের ভাষায় লিখুন—**ভার্সন কন্ট্রোল কী এবং কেন দরকার?**
2. Git এবং GitHub এর মধ্যে পার্থক্য লিখুন।
3. ভাবুন, যদি আপনার কম্পিউটারে Git ইনস্টল থাকে, আপনি কোন প্রজেক্টে প্রথমে ব্যবহার করতে চান? কেন?

অধ্যায় ২: Git এর বেসিক

2.1 Git কীভাবে কাজ করে?

Git আসলে একটি সিস্টেম, যা আপনার প্রজেক্টের ফাইলগুলোর পরিবর্তন ট্র্যাক করে। এটি তিনটি প্রধান ধাপে কাজ করে:

1. **Working Directory** – এখানে আপনি আপনার ফাইল এডিট করেন।
2. **Staging Area** – git add করলে পরিবর্তিত ফাইলগুলো এখানে জমা হয়।
3. **Repository (Commit History)** – git commit করলে সেই পরিবর্তন ইতিহাসে স্থায়ীভাবে রেকর্ড হয়।

☞ এভাবে Git প্রতিটি পরিবর্তনের ইতিহাস রাখে, যেন সহজে পুরোনো অবস্থায় ফিরে যাওয়া যায়।

2.2 Git ইন্সটলেশন

Windows এ

1. git-scm.com থেকে **Git for Windows** ডাউনলোড করুন।
2. ইনস্টলার চালু করুন এবং ডিফল্ট অপশন রেখেই Next করুন।
3. ইনস্টল শেষে Git Bash নামে একটি প্রোগ্রাম পাবেন।

Linux এ

```
sudo apt install git -y     # Ubuntu/Debian
sudo dnf install git -y     # Fedora
```

Mac এ

```
brew install git
```

ইনস্টলেশন যাচাই

```
git --version
```

2.3 Git কনফিগারেশন (প্রথমবার সেটআপ)

Git ব্যবহারের আগে আপনার নাম ও ইমেইল সেট করতে হবে।

```
git config --global user.name "আপনার নাম"
git config --global user.email "আপনার ইমেইল"
```

☞ এগুলো কমিট ইতিহাসে দেখা যাবে, যেন বোঝা যায় কে পরিবর্তন করেছে।

সব কনফিগারেশন দেখতে:

```
git config --list
```

2.4 Repository তৈরি

Git এ কাজ করার জন্য **Repository (রিপোজিটরি)** বানাতে হয়।

নতুন রিপোজিটরি শুরু করা

```
git init
```

☞ এই কমান্ড দিলে .git নামের একটি হিডেন ফোল্ডার তৈরি হবে যেখানে ইতিহাস সংরক্ষণ হবে।

বিদ্যমান প্রজেক্ট ক্লোন করা

```
git clone <repository-url>
```

☞ এটি GitHub বা অন্য কারো প্রজেক্ট কপি করে এনে আপনার কম্পিউটারে দেবে।

2.5 ফাইল ট্র্যাক করা

ফাইলের অবস্থা দেখা

```
git status
```

নতুন ফাইল যোগ করা (Staging এ নেওয়া)

```
git add filename.txt
```

সব ফাইল একসাথে যোগ করা

```
git add .
```

কমিট করা (ইতিহাসে সংরক্ষণ)

```
git commit -m "প্রথম কমিট"
```

☞ প্রতিটি কমিটে একটি **commit message** থাকে, যা সংক্ষেপে পরিবর্তনের বিবরণ দেয়।

2.6 ইতিহাস দেখা

সম্পূর্ণ লগ দেখা

```
git log
```

সংক্ষিপ্ত লগ দেখা

```
git log --oneline
```

ফাইলের পার্থক্য দেখা

```
git diff
```

2.7 অনুশীলনী

1. আপনার কম্পিউটারে Git ইন্সটল করুন।
2. একটি নতুন ফোল্ডার তৈরি করুন এবং তার ভেতরে git init চালান।
3. একটি hello.txt ফাইল বানিয়ে তাতে কিছু লিখুন।
4. git add hello.txt এবং git commit -m "প্রথম ফাইল যোগ" করুন।
5. এখন git log চালিয়ে দেখুন—আপনার প্রথম কমিট ইতিহাসে সংরক্ষিত হয়েছে।

অধ্যায় ৩: Git এর প্রধান কমান্ডসমূহ

এই অধ্যায়ে আমরা Git-এর সবচেয়ে বেশি ব্যবহৃত কমান্ডগুলো শিখবো। এগুলো ভালোভাবে আয়ত্ত করতে পারলে Git ব্যবহার অনেক সহজ হয়ে যাবে।

3.1 git status – বর্তমান অবস্থা দেখা

git status কমান্ড দিয়ে আপনি জানতে পারবেন আপনার রিপোজিটরির অবস্থা:

- কোন ফাইল পরিবর্তন হয়েছে,
- কোন ফাইল এখনো ট্র্যাক করা হয়নি,
- কোন ফাইল staging এ আছে।

```
git status
```

3.2 git add – ফাইলকে Staging এ নেওয়া

যখন আপনি কোনো ফাইল পরিবর্তন করবেন, সেটি Git সরাসরি সংরক্ষণ করে না। আগে git add দিয়ে Staging এ নিতে হয়।

```
git add filename.txt
```

☞ সব ফাইল একসাথে যোগ করতে চাইলে:

```
git add .
```

3.3 git commit – পরিবর্তন সংরক্ষণ করা

Staging এ নেওয়া ফাইলগুলোকে স্থায়ীভাবে সংরক্ষণ করতে হয় git commit দিয়ে।

```
git commit -m "নতুন ফাইল যোগ করা হলো"
```

☞ এখানে -m এর পরের অংশকে **commit message** বলা হয়। সবসময় স্পষ্ট ও ছোট করে লিখতে হবে।

3.4 git log – ইতিহাস দেখা

আপনার প্রজেক্টে যত কমিট হয়েছে তার তালিকা দেখতে পারবেন git log দিয়ে।

```
git log
```

☞ সংক্ষিপ্ত আকারে দেখতে:

```
git log --oneline
```

3.5 git diff – পার্থক্য দেখা

কোন ফাইলে কী পরিবর্তন হয়েছে তা দেখতে পারবেন git diff দিয়ে।

```
git diff
```

☞ এটি লাল ও সবুজ রঙে দেখাবে কোন অংশ মুছে গেছে আর কোন অংশ যোগ হয়েছে।

3.6 git branch – শাখা (Branch) ম্যানেজ করা

Branch হলো আলাদা লাইন যেখানে নতুন ফিচার বা কাজ করা যায় মূল প্রজেক্ট নষ্ট না করে।

নতুন branch তৈরি

```
git branch feature-1
```

সব branch দেখা

```
git branch
```

অন্য branch এ যাওয়া

```
git switch feature-1
```

3.7 git merge – branch একত্র করা

একটি branch এ করা কাজ মূল branch এ যোগ করতে হলে git merge ব্যবহার হয়।

```
git switch main
git merge feature-1
```

☞ এতে feature-1 এর কাজ main এ যোগ হয়ে যাবে।

3.8 git reset – ভুল ঠিক করা

যদি ভুল করে কোনো ফাইল যোগ করেন বা কমিট করেন, git reset দিয়ে সেটি ফিরিয়ে আনতে পারবেন।

Staging থেকে সরানো

```
git reset filename.txt
```

আগের কমিটে ফিরে যাওয়া

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

⚠️ সতর্কতা: --hard ব্যবহার করলে পরিবর্তিত ফাইল হারিয়ে যেতে পারে।

3.9 git rm – ফাইল মুছে ফেলা

Git এর ভেতর থেকে কোনো ফাইল মুছতে হলে ব্যবহার করুন:

```
git rm filename.txt
git commit -m "ফাইল মুছে ফেলা হলো"
```

3.10 অনুশীলনী

1. একটি নতুন রিপোজিটরি বানান।
2. তিনটি ফাইল তৈরি করুন: a.txt, b.txt, c.txt।
3. git add এবং git commit করে ফাইলগুলো সংরক্ষণ করুন।
4. একটি নতুন branch বানান test-branch নামে এবং তাতে কিছু পরিবর্তন করুন।
5. git merge দিয়ে কাজগুলো আবার মূল branch এ আনুন।
6. git log এবং git diff চালিয়ে পরিবর্তনগুলো পর্যবেক্ষণ করুন।

অধ্যায় ৪: GitHub পরিচিতি

GitHub হলো একটি অনলাইন প্ল্যাটফর্ম যেখানে আপনি আপনার Git রিপোজিটরি (repository) রাখতে পারেন। এটি কেবল ফাইল সংরক্ষণ করে না, বরং টিমওয়ার্ক, সহযোগিতা এবং ওপেন সোর্স কন্ট্রিবিউশন অনেক সহজ করে তোলে।

4.1 GitHub কী?

- Git হলো একটি টুল (আপনার কম্পিউটারে চলে),
- আর GitHub হলো একটি ওয়েবসাইট (অনলাইনে কাজ করার জায়গা)।

GitHub-এ রিপোজিটরি রাখলে:

- অন্যদের সাথে শেয়ার করা যায়,
- টিম মেম্বাররা একই সাথে কাজ করতে পারে,
- অনলাইনে ব্যাকআপ থাকে।

☞ সহজ ভাষায়: Git আপনার **কপি**, GitHub হলো **ক্লাউড স্টোরেজ**।

4.2 GitHub অ্যাকাউন্ট খোলা

1. github.com এ যান।
2. **Sign up** এ ক্লিক করুন।
3. ইউজারনেম, ইমেইল, এবং পাসওয়ার্ড দিন।
4. ইমেইল ভেরিফিকেশন করুন।
5. লগইন করলে আপনি নিজের ড্যাশবোর্ড দেখতে পাবেন।

4.3 নতুন Repository তৈরি করা

1. ড্যাশবোর্ড থেকে **New Repository** ক্লিক করুন।
2. রিপোজিটরির নাম দিন (যেমন: my-first-project)।
3. বর্ণনা (description) চাইলে লিখুন।
4. রিপোজিটরিটি **Public** বা **Private** হবে সেটি ঠিক করুন।
    - Public → সবাই দেখতে পারবে।
    - Private → কেবল আপনি ও আপনার টিম দেখতে পারবে।
5. চাইলে একটি README.md ফাইল তৈরি করে নিন।
6. **Create Repository** এ ক্লিক করুন।

☞ এবার আপনার অনলাইন রিপোজিটরি তৈরি হয়ে গেল।

4.4 GitHub এ ফাইল আপলোড করা

লোকাল কম্পিউটার থেকে ফাইল Push করা

প্রথমে লোকাল প্রজেক্টকে GitHub এর সাথে যুক্ত করতে হবে:

```
git remote add origin <repository-url>
git branch -M main
git push -u origin main
```

☞ এর পর থেকে git push কমান্ড দিলেই লোকাল পরিবর্তন GitHub এ চলে যাবে।

4.5 Repository ক্লোন করা

অন্য কারো প্রজেক্ট নিজের কম্পিউটারে আনতে চাইলে git clone ব্যবহার করতে হয়:

```
git clone <repository-url>
```

☞ এতে পুরো প্রজেক্ট আপনার কম্পিউটারে কপি হয়ে যাবে।

4.6 SSH Key সেটআপ (সহজভাবে)

প্রতিবার GitHub এ push করার সময় ইউজারনেম-পাসওয়ার্ড না দিয়ে SSH Key ব্যবহার করা যায়।

1. একটি SSH key তৈরি করুন:
2. ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
3. Key ফাইল ~/.ssh/id_rsa.pub থেকে কপি করুন।
4. GitHub > Settings > SSH and GPG keys > New SSH key এ গিয়ে পেস্ট করুন।
5. এখন থেকে push/pull করার সময় আর পাসওয়ার্ড দিতে হবে না।

4.7 GitHub Desktop (Optional)

যারা কমান্ড লাইন ব্যবহার করতে স্বাচ্ছন্দ্যবোধ করেন না, তারা **GitHub Desktop** নামের সফটওয়্যার ব্যবহার করতে পারেন।

- এটি গ্রাফিকাল ইন্টারফেস দিয়ে কাজ করা সহজ করে।
- commit, push, pull ইত্যাদি এক ক্লিকেই করা যায়।

4.8 অনুশীলনী

1. একটি GitHub অ্যাকাউন্ট খুলুন।
2. একটি নতুন রিপোজিটরি তৈরি করুন।
3. আপনার কম্পিউটারের একটি ছোট প্রজেক্ট সেই রিপোজিটরিতে push করুন।
4. অন্য কারো একটি Public রিপোজিটরি clone করে নিজের কম্পিউটারে নিন।
5. চেষ্টা করুন SSH key ব্যবহার করে push করতে।

অধ্যায় ৫: GitHub Workflow

GitHub শুধু ফাইল রাখার জায়গা নয়, এটি টিমওয়ার্ককে সহজ করার জন্য অনেক টুল সরবরাহ করে।
এই অধ্যায়ে আমরা শিখবো—**লোকাল প্রজেক্টকে GitHub এ আপলোড করা, অন্যের কাজ ডাউনলোড করা, Pull Request পাঠানো, এবং Fork ব্যবহার**।

5.1 লোকাল প্রজেক্ট GitHub এ পাঠানো (Push)

যখন আপনি নিজের কম্পিউটারে Git ব্যবহার করে কাজ করছেন, সেটি GitHub রিপোজিটরির সাথে যুক্ত করা যায়।

1. প্রথমে GitHub এ একটি repository তৈরি করুন (যেমন: my-first-repo)।
2. আপনার প্রজেক্ট ফোল্ডারে গিয়ে নিচের কমান্ড চালান:

```
git remote add origin <repository-url>
git branch -M main
git push -u origin main
```

☞ এখন থেকে যখনই পরিবর্তন করবেন, শুধু লিখবেন:

```
git add .
git commit -m "পরিবর্তন করা হলো"
git push
```

5.2 GitHub থেকে প্রজেক্ট আনা (Pull এবং Fetch)

যদি টিমমেটরা GitHub এ পরিবর্তন করে, তাহলে সেগুলো আপনার কম্পিউটারে আনার জন্য ব্যবহার করতে হয়:

পরিবর্তন ডাউনলোড এবং সাথে সাথে মিশিয়ে দেওয়া (Pull)

```
git pull origin main
```

শুধু ডাউনলোড করা (Fetch)

```
git fetch origin
```

☞ Fetch করলে পরিবর্তনগুলো ডাউনলোড হবে, কিন্তু সাথে সাথে মিশবে না।

5.3 Fork বনাম Clone

Clone

- আপনি সরাসরি অন্যের রিপোজিটরি নিজের কম্পিউটারে কপি করেন।
- সাধারণত ব্যক্তিগত কাজ বা শেখার জন্য ব্যবহার হয়।

Fork

- এটি GitHub এর একটি ফিচার।
- অন্যের রিপোজিটরির একটি **কপি আপনার GitHub অ্যাকাউন্টে** চলে আসে।
- এরপর আপনি নিজের মতো পরিবর্তন করতে পারবেন এবং চাইলে মূল প্রজেক্টে Pull Request পাঠাতে পারবেন।

☞ **উদাহরণ**: যদি কোনো ওপেন সোর্স প্রজেক্টে কন্ট্রিবিউট করতে চান, প্রথমে Fork করতে হয়।

5.4 Pull Request (PR)

Pull Request হলো এমন একটি প্রক্রিয়া, যেখানে আপনি মূল প্রজেক্টে আপনার পরিবর্তন যোগ করার অনুরোধ পাঠান।

Pull Request পাঠানোর ধাপ:

1. প্রজেক্টটিকে Fork করুন।
2. আপনার Fork করা রিপোজিটরি ক্লোন করে লোকালি পরিবর্তন করুন।
3. পরিবর্তন commit এবং push করুন আপনার GitHub রিপোজিটরিতে।
4. মূল প্রজেক্টের পেজে গিয়ে **New Pull Request** ক্লিক করুন।
5. আপনার পরিবর্তন বর্ণনা করে পাঠান।

☞ প্রজেক্টের মেইনটেইনার যদি আপনার পরিবর্তন ভালো মনে করেন, তিনি সেটি মূল প্রজেক্টে merge করবেন।

5.5 Issues এবং Discussions

GitHub এ প্রতিটি রিপোজিটরিতে **Issues** এবং **Discussions** থাকে:

- **Issues** → সমস্যা বা নতুন ফিচারের প্রস্তাব লিখতে হয়।
- **Discussions** → সাধারণ প্রশ্ন, টিপস, বা প্রজেক্ট নিয়ে আলাপচারিতা।

☞ এগুলো টিমওয়ার্কে যোগাযোগ সহজ করে।

5.6 Releases এবং Tags

যখন প্রজেক্টের একটি নির্দিষ্ট ভার্সন তৈরি হয় (যেমন v1.0, v2.0), তখন সেটিকে **Tag** দিয়ে চিহ্নিত করা যায়।

```
git tag v1.0
git push origin v1.0
```

☞ GitHub এ এগুলোকে **Release** হিসেবে প্রকাশ করা যায়, যাতে ব্যবহারকারীরা নির্দিষ্ট ভার্সন ডাউনলোড করতে পারে।

5.7 অনুশীলনী

1. নিজের কম্পিউটার থেকে একটি প্রজেক্ট GitHub এ push করুন।
2. অন্যের একটি public রিপোজিটরি clone করে আপনার কম্পিউটারে নিন।
3. একটি ওপেন সোর্স প্রজেক্ট fork করুন এবং নিজের GitHub এ কপি তৈরি করুন।
4. সেই fork করা প্রজেক্টে একটি ছোট পরিবর্তন করে Pull Request পাঠান।
5. একটি টেস্ট রিপোজিটরিতে tag দিয়ে v1.0 নামে একটি release তৈরি করুন।

অধ্যায় ৬: টিমওয়ার্ক এবং কলাবোরেশন

Git এবং GitHub এর সবচেয়ে বড় শক্তি হলো—এটি একসাথে অনেকজনকে একই প্রজেক্টে কাজ করার সুযোগ দেয়।
যেমন, একটি সফটওয়্যার প্রজেক্টে একজন কাজ করছে লগইন সিস্টেমে, আরেকজন কাজ করছে ডিজাইন-এ, আবার অন্য কেউ কাজ করছে ডাটাবেসে।
সবাই একসাথে কাজ করলেও Git-এর মাধ্যমে এগুলো সুন্দরভাবে ম্যানেজ করা সম্ভব।

6.1 টিমওয়ার্কে Git এর ভূমিকা

- **একই প্রজেক্টে অনেকজন একসাথে কাজ করতে পারে** → প্রত্যেকের কাছে পুরো কোডবেস থাকে।
- **প্রতিটি পরিবর্তন ট্র্যাক হয়** → কে, কখন, কী পরিবর্তন করেছে সব রেকর্ড থাকে।
- **Branch ব্যবহার করে আলাদা কাজ করা যায়** → নতুন ফিচার তৈরি, বাগ ফিক্স ইত্যাদি।
- **GitHub এ কোড শেয়ার করা যায়** → ফলে টিমের সবাই সহজে সহযোগিতা করতে পারে।

6.2 Merge Conflict কী?

যখন দুইজন ডেভেলপার একই ফাইলে একই জায়গায় পরিবর্তন করে, তখন **Merge Conflict** হয়।

Git নিজে থেকে বুঝতে পারে না কোন পরিবর্তন রাখতে হবে, তাই ব্যবহারকারীকে ম্যানুয়ালি ঠিক করতে হয়।

উদাহরণ:

- আপনি index.html ফাইলে শিরোনাম লিখলেন:
- <h1>Welcome to My Website</h1>
- আপনার সহকর্মী একই লাইনে লিখলেন:
- <h1>আমার ওয়েবসাইটে স্বাগতম</h1>

☞ এখন যখন merge করবেন, Git কনফ্লিক্ট দেখাবে।

সমাধান:

1. Git কনফ্লিক্ট মার্ক করে দেয় ফাইলের মধ্যে:
2. <<<<<<< HEAD
3. <h1>Welcome to My Website</h1>
4. =======
5. <h1>আমার ওয়েবসাইটে স্বাগতম</h1>
6. >>>>>>> branch-এ পরিবর্তন
7. আপনি সিদ্ধান্ত নেবেন কোন অংশ রাখবেন (বা দুটো একসাথে মিলাবেন)।
8. ফাইল সেভ করে আবার commit করবেন।

6.3 Merge Conflict এড়ানোর উপায়

- বারবার **pull** করুন, যাতে টিমের সর্বশেষ কোড আপনার কাছে থাকে।
- একই ফাইলে একসাথে কাজ না করার চেষ্টা করুন।
- বড় টিমে কাজ হলে ফাইল বা ফিচার ভাগ করে নিন।
- ছোট এবং স্পষ্ট commit করুন, যাতে সহজে বোঝা যায়।

6.4 Code Review কী?

Code Review হলো এমন একটি প্রক্রিয়া, যেখানে একজন ডেভেলপারের লেখা কোড অন্য কেউ পর্যালোচনা করে।

GitHub এ Code Review করার ধাপ:

1. ডেভেলপার কোড লিখে একটি **Pull Request** পাঠাবে।
2. টিম মেম্বাররা PR খুলে কোড দেখবে।
3. যদি কোনো সমস্যা থাকে, রিভিউয়ার কমেন্ট দেবে।
4. সব সমস্যা সমাধান হলে PR merge করা হবে।

☞ এর ফলে কোডের মান উন্নত হয় এবং ভুল কমে যায়।

6.5 Branch Protection Rules

টিমওয়ার্কে কখনো কখনো ভুল করে কেউ সরাসরি মূল branch (main/master) এ push করে দিতে পারে।
এটি এড়াতে **Branch Protection Rules** ব্যবহার করা হয়।

সুবিধা:

- কেউ সরাসরি main এ push করতে পারবে না।
- PR (Pull Request) না করলে কোড merge হবে না।
- নির্দিষ্ট টিম মেম্বারের অনুমতি ছাড়া merge করা যাবে না।

☞ Branch Protection Rules সেট করতে:

1. GitHub রিপোজিটরির Settings > Branches এ যান।

2. **Add rule** এ ক্লিক করুন।
3. main branch নির্বাচন করুন।
4. Protect করুন যাতে শুধুমাত্র PR এর মাধ্যমে পরিবর্তন আসতে পারে।

6.6 টিমওয়ার্কে ভালো প্র্যাকটিস

- ছোট ও পরিষ্কার commit message লিখুন।
- feature অনুযায়ী branch ব্যবহার করুন (যেমন: feature/login, bugfix/ui)।
- বারবার pull করুন যাতে সর্বশেষ কোড থাকে।
- বড় টিমে কাজ করলে কোড review বাধ্যতামূলক করুন।
- PR merge করার আগে টেস্ট করে নিন।

6.7 অনুশীলনী

1. দুইটি আলাদা branch তৈরি করুন: feature1 এবং feature2।
2. একই ফাইলে একই লাইনে আলাদা পরিবর্তন করুন।
3. merge করার সময় conflict তৈরি করুন এবং নিজে সমাধান করুন।
4. GitHub এ একটি PR তৈরি করুন এবং অন্য কারো মাধ্যমে Code Review করান।
5. Branch Protection Rule সেট করুন যাতে main branch এ সরাসরি push করা না যায়।

অধ্যায় ৭: অ্যাডভান্সড Git

আগের অধ্যায়গুলোতে আমরা Git-এর মৌলিক ব্যবহার শিখেছি—add, commit, branch, merge ইত্যাদি।
কিন্তু বাস্তব প্রজেক্টে অনেক জটিল পরিস্থিতি আসে। সেসব ক্ষেত্রে কিছু উন্নত Git কমান্ড আমাদের কাজকে অনেক সহজ করে।

7.1 Rebase বনাম Merge

Merge কী?

যখন আপনি একটি branch-এর কাজ মূল branch (main)-এ যোগ করতে চান, তখন git merge ব্যবহার করেন। এতে একটি **merge commit** তৈরি হয়।

```
git switch main
git merge feature-branch
```

☞ Merge করলে history একটু জটিল হয়ে যায়, কারণ আলাদা commit tree থেকে যায়।

Rebase কী?

Rebase হলো অন্য branch-এর commit-গুলোকে নিয়ে এসে **আপনার branch-এর উপরে বসানো**।

```
git switch feature-branch
git rebase main
```

☞ এতে history সোজা ও পরিষ্কার থাকে (linear history)।

তুলনা:

- Merge = ইতিহাস অক্ষত রাখে, কিন্তু জটিল হতে পারে।
- Rebase = ইতিহাস সোজা হয়, কিন্তু commit ID বদলে যায়।

7.2 Git Stash

কখনো এমন হয় যে আপনি কাজের মাঝপথে আছেন, কিন্তু commit করতে চান না।
সেই পরিবর্তনগুলো অস্থায়ীভাবে রেখে পরে আবার ফিরিয়ে আনার জন্য git stash ব্যবহার হয়।

পরিবর্তন Stash করা

```
git stash
```

Stash তালিকা দেখা

```
git stash list
```

Stash ফিরিয়ে আনা

```
git stash apply
```

Stash মুছে ফেলা

```
git stash drop
```

☞ এটি বিশেষ করে কাজে লাগে যখন হঠাৎ branch বদলাতে হয়।

7.3 Git Cherry-pick

Cherry-pick মানে হলো ইতিহাস থেকে নির্দিষ্ট একটি commit তুলে এনে অন্য branch-এ ব্যবহার করা।

```
git cherry-pick <commit-id>
```

☞ এটি কাজে লাগে যদি কোনো bug fix একটি branch-এ করা হয়, কিন্তু সেই fix অন্য branch-এও দরকার হয়।

7.4 Git Submodules

কখনো একটি প্রজেক্টের মধ্যে আরেকটি প্রজেক্ট যুক্ত করতে হয়। যেমন—

- মূল প্রজেক্টে আপনি একটি বাইরের লাইব্রেরি ব্যবহার করছেন।

Git-এ এ জন্য **submodule** ব্যবহার হয়।

Submodule যোগ করা

```
git submodule add <repo-url>
```

Submodule আপডেট করা

```
git submodule update --init --recursive
```

☞ এটি dependency ম্যানেজ করতে কাজে লাগে।

7.5 Git Hooks

Git-এ কিছু কাজ স্বয়ংক্রিয়ভাবে করানো যায় **hooks** ব্যবহার করে।
Hooks হলো স্ক্রিপ্ট যা Git-এর নির্দিষ্ট ইভেন্টে (যেমন commit, push) রান হয়।

উদাহরণ:

- **pre-commit hook** → commit করার আগে কোড lint/check করা।
- **pre-push hook** → push করার আগে টেস্ট রান করানো।

☞ Hooks .git/hooks ফোল্ডারে থাকে।
আপনি সেখানে নিজের স্ক্রিপ্ট লিখে অটোমেশন করতে পারবেন।

7.6 উন্নত Git ব্যবহার করার সময় সতর্কতা

- Rebase করার আগে সবসময় pull করে নিন, না হলে conflict বেড়ে যেতে পারে।
- Stash ব্যবহার করলে পরে ভুলে না যান—নাহলে পরিবর্তন হারিয়ে যাবে।
- Cherry-pick করার সময় commit ID সঠিক কিনা নিশ্চিত হোন।
- Submodules আপডেট করা না হলে dependency mismatch হতে পারে।
- Hooks ব্যবহার করলে টিমের সবাই যেন সেটি মেনে চলে তা নিশ্চিত করতে হবে।

7.7 অনুশীলনী

1. একটি branch তৈরি করুন এবং git rebase ব্যবহার করে main branch-এর সাথে merge করুন।
2. কিছু অসমাপ্ত পরিবর্তন করুন এবং git stash দিয়ে সেভ করুন। পরে git stash apply দিয়ে ফিরিয়ে আনুন।
3. একটি কমিট ID নিয়ে সেটি অন্য branch-এ git cherry-pick করুন।
4. একটি ছোট প্রজেক্টকে submodule হিসেবে যুক্ত করুন।
5. একটি pre-commit hook বানান, যা কমিট করার আগে টার্মিনালে “Commit হচ্ছে...” লিখে দেখাবে।

অধ্যায় ৮: GitHub এর বাড়তি ফিচার

GitHub শুধু কোড রাখার জায়গা নয়, বরং সফটওয়্যার ডেভেলপমেন্টকে আরও সহজ এবং কার্যকর করার জন্য অনেক বিশেষ ফিচার দেয়।
এই অধ্যায়ে আমরা চারটি জনপ্রিয় ফিচার শিখবো:

- GitHub Actions (CI/CD)
- GitHub Pages (ওয়েবসাইট হোস্টিং)
- GitHub Projects (কাজ ম্যানেজমেন্ট)
- GitHub Codespaces (অনলাইন কোডিং এনভায়রনমেন্ট)

8.1 GitHub Actions (CI/CD)

CI/CD কী?

- **CI (Continuous Integration)** → কোডে পরিবর্তন হলে স্বয়ংক্রিয়ভাবে build/test চালানো।
- **CD (Continuous Deployment/Delivery)** → কোড টেস্ট পাস করলে সেটি স্বয়ংক্রিয়ভাবে সার্ভারে ডিপ্লয় করা।

GitHub Actions কীভাবে কাজ করে?

- .github/workflows ফোল্ডারের মধ্যে YAML ফাইল লিখে workflow তৈরি করা হয়।
- যেমন—প্রতিবার কেউ কোড push করলে স্বয়ংক্রিয়ভাবে টেস্ট রান হবে।

উদাহরণ (Node.js প্রজেক্টে টেস্ট):

```
name: Node.js CI
```

```
on: [push]

jobs:
  build:
    runs-on: ubuntu-latest
    steps:
    - uses: actions/checkout@v3
    - name: Use Node.js
      uses: actions/setup-node@v3
      with:
        node-version: '16'
    - run: npm install
    - run: npm test
```

☞ এর মাধ্যমে আপনি আপনার প্রজেক্টের গুণগত মান বজায় রাখতে পারবেন।

8.2 GitHub Pages (ওয়েবসাইট হোস্টিং)

GitHub Pages হলো একটি ফ্রি ওয়েবসাইট হোস্টিং সার্ভিস, যা সরাসরি আপনার রিপোজিটরি থেকে ওয়েবসাইট তৈরি করে।

ব্যবহার করার ধাপ:

1. একটি রিপোজিটরি তৈরি করুন।
2. তাতে index.html ফাইল রাখুন।
3. Settings > Pages এ যান।
4. কোন branch থেকে ওয়েবসাইট তৈরি হবে সেটি নির্বাচন করুন (সাধারণত main) I
5. Save করলে একটি URL পাবেন, যেমন:
6. https://username.github.io/my-website/

☞ অনেক ডেভেলপার তাদের portfolio, project documentation বা blog GitHub Pages দিয়ে হোস্ট করেন।

8.3 GitHub Projects (কাজ ম্যানেজমেন্ট)

GitHub Projects হলো একটি **Kanban-style project board**, যা দিয়ে টিমের কাজ ম্যানেজ করা যায়।

কীভাবে কাজ করে?

- আপনি একটি প্রজেক্ট তৈরি করতে পারেন “To Do”, “In Progress”, “Done” কলাম দিয়ে।
- Issues বা Pull Requests কার্ড আকারে এই বোর্ডে যোগ করা যায়।
- টিমের সবাই দেখতে পারে কে কী কাজ করছে।

☞ এটি Trello বা Jira এর মতো, কিন্তু সরাসরি GitHub এর সাথে ইন্টিগ্রেটেড।

8.4 GitHub Codespaces (অনলাইন কোডিং এনভায়রনমেন্ট)

GitHub Codespaces হলো একটি অনলাইন IDE (Integrated Development Environment), যা আপনার ব্রাউজারেই কোড চালাতে দেয়।

সুবিধা:

- VS Code-এর মতো ইন্টারফেস।
- সরাসরি GitHub থেকে কোড ওপেন করে রান করা যায়।
- কোনো সেটআপ ছাড়াই ক্লাউডে কাজ শুরু করা যায়।

☞ এর মানে হলো, আপনি যেকোনো জায়গা থেকে শুধু ব্রাউজার দিয়ে কোড লিখতে পারবেন।

8.5 সারসংক্ষেপ

- **GitHub Actions** → স্বয়ংক্রিয় টেস্ট ও ডিপ্লয়মেন্ট।
- **GitHub Pages** → ফ্রি ওয়েবসাইট হোস্টিং।
- **GitHub Projects** → টিমের কাজ ম্যানেজমেন্ট।
- **GitHub Codespaces** → অনলাইন কোডিং এনভায়রনমেন্ট।

8.6 অনুশীলনী

1. একটি রিপোজিটরিতে একটি ছোট workflow লিখুন যা প্রতিবার push করলে “Hello GitHub Actions” প্রিন্ট করে।
2. একটি index.html ফাইল বানিয়ে GitHub Pages দিয়ে ফ্রি ওয়েবসাইট হোস্ট করুন।
3. একটি ছোট টিম প্রজেক্ট তৈরি করে GitHub Projects-এ কাজ ভাগ করে নিন।
4. Codespaces এ গিয়ে একটি রিপোজিটরি ওপেন করুন এবং সরাসরি ব্রাউজার থেকে কোড এডিট করুন।

অধ্যায় ৯: বাস্তব উদাহরণ (Case Studies)

এখন পর্যন্ত আমরা Git এবং GitHub এর কমান্ড, ফিচার ও ওয়ার্কফ্লো শিখেছি।
কিন্তু বাস্তবে কিভাবে এগুলো ব্যবহার হয় সেটাই সবচেয়ে গুরুত্বপূর্ণ।
এই অধ্যায়ে আমরা তিনটি ভিন্ন উদাহরণ দেখব—

1. একক প্রজেক্ট (Single Developer Workflow)
2. টিম প্রজেক্ট (Team Collaboration Workflow)
3. ওপেন সোর্স প্রজেক্টে কন্ট্রিবিউশন

9.1 একক প্রজেক্টে Git ব্যবহার

পরিস্থিতি

আপনি একজন শিক্ষার্থী/ডেভেলপার। আপনার একটি ব্যক্তিগত প্রজেক্ট আছে—যেমন **একটি ব্লগ ওয়েবসাইট**।

Workflow

1. একটি লোকাল রিপোজিটরি তৈরি করুন:
2. git init
3. প্রতিদিন কিছু ফিচার যোগ করুন এবং commit করুন:
4. git add .
5. git commit -m "হোমপেজ ডিজাইন সম্পন্ন"
6. কাজের ব্যাকআপ রাখতে GitHub-এ push করুন:
7. git remote add origin <repo-url>
8. git push -u origin main

9. একটি নতুন ফিচার বানাতে চাইলে আলাদা branch ব্যবহার করুন:
10. git branch feature-comments
11. git switch feature-comments
12. কাজ শেষ হলে main branch-এ merge করুন।

☞ এভাবে আপনার পুরো প্রজেক্টের ইতিহাস নিরাপদে GitHub এ থাকবে।

9.2 টিম প্রজেক্টে Git ব্যবহার

পরিস্থিতি

ধরুন আপনার একটি সফটওয়্যার ডেভেলপমেন্ট টিম আছে, যেখানে ৪ জন ডেভেলপার কাজ করছে একটি **ই-কমার্স ওয়েবসাইটে**।

Workflow

1. **Repository Setup** → একজন টিম লিডার GitHub এ রিপোজিটরি তৈরি করে সবাইকে collaborator হিসেবে যোগ করবে।
2. **Branching Strategy** →
    - main branch = স্থিতিশীল কোড
    - dev branch = নতুন ফিচার merge করার জন্য
    - feature branches = প্রতিটি ফিচারের জন্য আলাদা branch
3. **কাজের ধাপ** →
    - ডেভেলপার A → feature-login branch
    - ডেভেলপার B → feature-cart branch
    - ডেভেলপার C → feature-payment branch
    - ডেভেলপার D → feature-dashboard branch
4. **Pull Request (PR)** → প্রত্যেকে কাজ শেষ হলে dev branch-এ PR পাঠাবে।

5. **Code Review** → টিম লিডার PR রিভিউ করবে এবং সমস্যা থাকলে কমেন্ট দেবে।
6. **Merge** → সব ঠিক থাকলে dev branch-এ merge হবে।
7. **Release** → একটি নির্দিষ্ট ভার্সন প্রস্তুত হলে dev branch → main branch-এ merge করে release তৈরি হবে।

☞ এই প্রক্রিয়ায় সবাই একসাথে কাজ করতে পারে কোনো ফাইল নষ্ট না করে।

9.3 ওপেন সোর্স প্রজেক্টে কন্ট্রিবিউশন

পরিস্থিতি

আপনি GitHub এ একটি জনপ্রিয় ওপেন সোর্স প্রজেক্ট খুঁজে পেয়েছেন (যেমন React, Django বা Laravel)। আপনি তাতে কন্ট্রিবিউট করতে চান।

Workflow

1. প্রজেক্টটি GitHub থেকে **Fork** করুন।
2. আপনার GitHub অ্যাকাউন্টে ওই প্রজেক্টের একটি কপি তৈরি হবে।
3. Fork করা রিপোজিটরি **clone** করে লোকালি নিন:
4. git clone <your-fork-url>
5. একটি নতুন branch তৈরি করুন:
6. git switch -c fix-typo
7. পরিবর্তন করুন (যেমন একটি bug ফিক্স বা typo ঠিক করা)।
8. commit এবং push করুন আপনার GitHub রিপোজিটরিতে।
9. মূল প্রজেক্টের পেজে গিয়ে **Pull Request (PR)** পাঠান।
10. প্রজেক্ট মেইনটেইনার যদি পরিবর্তন অনুমোদন করেন, তাহলে আপনার কোড মূল প্রজেক্টে merge হবে।

☞ এভাবেই সারা বিশ্বের মানুষ ওপেন সোর্স প্রজেক্টে অবদান রাখে।

9.4 সারসংক্ষেপ

- একক প্রজেক্টে Git ব্যবহার করলে ইতিহাস নিরাপদ থাকে এবং ফিচার আলাদা branch এ করা যায়।
- টিম প্রজেক্টে branch strategy, PR এবং code review খুব গুরুত্বপূর্ণ।
- ওপেন সোর্স কন্ট্রিবিউশনের জন্য Fork, Clone এবং Pull Request শিখতে হয়।

9.5 অনুশীলনী

1. একটি ছোট একক প্রজেক্ট শুরু করে প্রতিদিন কমপক্ষে ৩টি commit করুন।
2. একটি বন্ধু/সহকর্মীকে নিয়ে GitHub এ একটি টেস্ট রিপোজিটরি বানান এবং দুজন একসাথে কাজ করুন।
3. GitHub থেকে একটি ওপেন সোর্স প্রজেক্ট Fork করুন এবং অন্তত একটি PR পাঠান।

অধ্যায় ১০: টিপস ও বেস্ট প্র্যাকটিস

Git এবং GitHub ব্যবহার করার সময় শুধু কমান্ড জানলেই হবে না, কিছু ভালো অভ্যাস (best practices) অনুসরণ করতে হয়।
এগুলো অনুসরণ করলে আপনার প্রজেক্ট হবে **পরিষ্কার, সুশৃঙ্খল এবং নিরাপদ**।

10.1 ভালো Commit Message লেখার নিয়ম

Commit message হলো প্রজেক্টের ইতিহাসের অংশ। এটি পরিষ্কার না হলে পরবর্তীতে বোঝা কঠিন হয়।

ভালো commit message এর বৈশিষ্ট্য:

- ছোট ও স্পষ্ট হবে (৫০-৭২ অক্ষরের মধ্যে)।
- বর্তমান কালে লেখা হবে (যেমন: "Fix bug", "Add login page")।
- প্রতিটি কমিট একটি নির্দিষ্ট পরিবর্তনের জন্য হবে।

উদাহরণ:

✅ ভালো:

- Add user authentication system
- Fix typo in README
- Update payment API integration

❌ খারাপ:

- updated code
- fix
- done

☞ মনে রাখবেন, commit message ভবিষ্যতের আপনারই জন্য সহায়ক।

10.2 Branch Naming Convention

প্রতিটি branch-এর নাম পরিষ্কার এবং অর্থবহ হতে হবে।

সাধারণ কনভেনশন:

- **feature/** → নতুন ফিচার যোগ করার জন্য
  - feature/login-system
- **bugfix/** → বাগ ঠিক করার জন্য
  - bugfix/cart-total
- **hotfix/** → জরুরি সমস্যা সমাধানের জন্য
  - hotfix/payment-crash
- **release/** → নির্দিষ্ট রিলিজ প্রস্তুতের জন্য
  - release/v1.0

☞ এর ফলে টিমে কাজ করার সময় সহজেই বোঝা যায় কোন branch কীসের জন্য।

10.3 Tagging এবং Release Strategy

Tag ব্যবহার করে প্রজেক্টের একটি নির্দিষ্ট অবস্থান চিহ্নিত করা যায়।
সাধারণত release version এর জন্য ব্যবহার করা হয়।

উদাহরণ:

```
git tag v1.0
git push origin v1.0
```

সাধারণ version naming convention (Semantic Versioning):

- **v1.0.0** → Major release (বড় পরিবর্তন)
- **v1.1.0** → Minor release (ছোট নতুন ফিচার)
- **v1.1.1** → Patch release (বাগ ফিক্স)

☞ GitHub-এ Release পেজ ব্যবহার করলে ব্যবহারকারীরা সহজে নির্দিষ্ট ভার্সন ডাউনলোড করতে পারে।

10.4 নিরাপত্তা (Security) বিষয়ক টিপস

- কখনো **পাসওয়ার্ড বা API key** রিপোজিটরিতে commit করবেন না।
- .gitignore ফাইল ব্যবহার করুন যেন সংবেদনশীল ফাইল ট্র্যাক না হয়।
- Private রিপোজিটরি ব্যবহার করুন যদি প্রজেক্ট ব্যক্তিগত হয়।
- টিম প্রজেক্টে **branch protection rules** ব্যবহার করুন।
- Two-Factor Authentication (2FA) চালু করুন GitHub অ্যাকাউন্টে।

10.5 পরিষ্কার Workflow বজায় রাখা

- প্রতিদিন কাজ শুরু করার আগে git pull দিয়ে সর্বশেষ কোড নিন।
- কাজ শেষ হলে ছোট ছোট commit করুন।
- Pull Request merge করার আগে সবসময় টেস্ট করুন।
- বড় পরিবর্তন একসাথে না করে ভাগ ভাগ করে commit করুন।
- পুরোনো branch আর দরকার না হলে delete করুন।

10.6 সারসংক্ষেপ

- ভালো commit message লেখুন → ইতিহাস পরিষ্কার থাকবে।
- branch naming convention অনুসরণ করুন → কাজের ধরন বোঝা সহজ হবে।
- release strategy ব্যবহার করুন → version ম্যানেজ করা সহজ হবে।
- নিরাপত্তা মেনে চলুন → প্রজেক্ট সুরক্ষিত থাকবে।
- পরিষ্কার workflow বজায় রাখুন → টিমওয়ার্ক সহজ হবে।

10.7 অনুশীলনী

1. একটি প্রজেক্টে অন্তত ৫টি commit করুন এবং ভালো commit message ব্যবহার করুন।
2. আলাদা আলাদা branch তৈরি করুন—একটি feature branch এবং একটি bugfix branch।
3. একটি test release বানান v1.0.0 নামে এবং GitHub-এ release তৈরি করুন।
4. আপনার প্রজেক্টে .gitignore ব্যবহার করে এমন ফাইল বাদ দিন যা commit হওয়া উচিত নয় (যেমন .env, node_modules)।

অধ্যায় ১১: FAQ / সমস্যার সমাধান

Git বা GitHub ব্যবহার করার সময় প্রায়ই কিছু সাধারণ সমস্যা দেখা দেয়।
এই অধ্যায়ে আমরা সেই সমস্যাগুলো, কারণ এবং সমাধান আলোচনা করব।

11.1 “detached HEAD” সমস্যা

কখন হয়?

যখন আপনি সরাসরি কোনো commit checkout করেন (branch নয়), তখন Git বলে: **detached HEAD**।

```
git checkout <commit-id>
```

👉 এতে আপনি commit দেখতে পারবেন, কিন্তু নতুন পরিবর্তন করলে সেগুলো কোনো branch-এ যুক্ত হবে না।

সমাধান

যদি সেই অবস্থায় কাজ রাখতে চান → একটি নতুন branch তৈরি করুন:

```
git switch -c new-branch
```

11.2 Merge Conflict

কখন হয়?

যখন দুইজন একই ফাইলের একই জায়গায় পরিবর্তন করেন, এবং merge করার সময় Git বুঝতে পারে না কোনটি রাখতে হবে।

সমাধান

1. git status চালিয়ে conflict ফাইল চিহ্নিত করুন।
2. ফাইল খুলে conflict মার্কার (<<<<<<<, =======, >>>>>>>) ঠিক করুন।
3. ঠিক করার পর আবার commit করুন:
4. git add filename
5. git commit -m "Merge conflict resolved"

11.3 ভুলে ফাইল যোগ হয়ে গেছে (Unstage file)

সমস্যা

আপনি ভুল করে git add দিয়ে staging area-তে ফাইল নিয়েছেন।

সমাধান

ফাইলটিকে unstaged করতে:

```
git reset filename.txt
```

11.4 ভুলে sensitive তথ্য commit হয়ে গেছে

সমস্যা

কখনো কখনো API key, পাসওয়ার্ড বা secret ফাইল commit হয়ে যায়।

সমাধান

1. ফাইল .gitignore এ যোগ করুন।

2. history থেকে মুছতে (বড় পদক্ষেপ, সতর্ক থাকতে হবে):
3. git filter-branch --force --index-filter \
4. "git rm --cached --ignore-unmatch filename.txt" \
5. --prune-empty --tag-name-filter cat -- --all
6. পরে force push করতে হবে:
7. git push origin --force --all

☞ সবচেয়ে ভালো সমাধান হলো শুরুতেই .gitignore ব্যবহার করা।

11.5 ভুল করে branch মুছে ফেলা

সমস্যা

লোকালি branch delete হয়ে গেছে।

সমাধান

Remote থেকে আবার আনার চেষ্টা করুন:

```
git fetch origin
git checkout -b branch-name origin/branch-name
```

☞ যদি লোকালি commit থাকে কিন্তু push করা না হয়, তবে recover কঠিন হতে পারে। তাই সবসময় সময়মতো push করুন।

11.6 git push কাজ করছে না

কারণ

- Remote URL ঠিকমতো সেট করা হয়নি।
- Branch নাম মেলেনি।

সমাধান

```
git remote -v
git remote set-url origin <repo-url>
git push -u origin main
```

11.7 Repository কে Public/Private করা

প্রশ্ন

আমি কীভাবে GitHub এ repository private থেকে public (বা উল্টো) করতে পারি?

উত্তর

1. GitHub এ repository খুলুন।
2. Settings > Danger Zone এ যান।
3. Change Visibility → Public/Private নির্বাচন করুন।

☞ সতর্কতা: Public করলে সবাই কোড দেখতে পাবে।

11.8 টার্মিনালে বাংলা/Unicode সমস্যা

সমস্যা

বাংলা বা Unicode commit message দিলে গার্বেজ দেখায়।

সমাধান

টার্মিনাল UTF-8 এ সেট করুন।

```
git config --global i18n.commitEncoding utf-8
git config --global i18n.logOutputEncoding utf-8
```

11.9 GitHub এ push করার সময় বারবার পাসওয়ার্ড চাই

কারণ

HTTPS ব্যবহার করছেন।

সমাধান

SSH Key সেটআপ করুন (অধ্যায় ৪.৬ দেখুন)।
অথবা GitHub CLI ব্যবহার করুন।

11.10 সারসংক্ষেপ

- detached HEAD → নতুন branch তৈরি করুন।
- Merge Conflict → ফাইল ঠিক করে commit করুন।
- ভুল add → git reset ব্যবহার করুন।
- sensitive তথ্য commit → .gitignore ব্যবহার করুন।
- branch delete → remote থেকে ফিরিয়ে আনুন।
- push error → remote URL ঠিক করুন।
- repo visibility → GitHub Settings থেকে পরিবর্তন করুন।
- Unicode সমস্যা → UTF-8 এ কনফিগার করুন।

- বারবার পাসওয়ার্ড → SSH Key ব্যবহার করুন।

11.11 অনুশীলনী

1. একটি রিপোজিটরি তৈরি করে ইচ্ছে করে Merge Conflict তৈরি করুন এবং সমাধান করুন।
2. একটি ফাইল ভুল করে git add করুন এবং পরে git reset দিয়ে unstaged করুন।
3. একটি test repository public থেকে private করুন।
4. একটি বাংলা commit message লিখুন এবং log-এ দেখুন সেটি সঠিকভাবে দেখা যায় কিনা।

অধ্যায় ১২: পরিশিষ্ট (Appendix)

এটি বইয়ের শেষ অধ্যায়। এখানে আমরা Git এবং GitHub ব্যবহারকারীদের জন্য একটি **Cheat Sheet (দ্রুত কমান্ড তালিকা)**, কিছু গুরুত্বপূর্ণ **রিসোর্স**, এবং একটি **Glossary (টার্মিনোলজি/শব্দকোষ বাংলায় অর্থসহ)** দেবো।

12.1 Git Cheat Sheet (সবচেয়ে বেশি ব্যবহৃত কমান্ড)

রিপোজিটরি সেটআপ

```
git init                          # নতুন রিপোজিটরি তৈরি
git clone <url>                   # বিদ্যমান রিপোজিটরি কপি
git config --global user.name "নাম"
git config --global user.email "ইমেইল"
```

ফাইল ম্যানেজমেন্ট

```
git status                        # অবস্থা দেখা
git add filename.txt              # ফাইল যোগ করা
git add .                         # সব ফাইল যোগ করা
git commit -m "বার্তা"            # পরিবর্তন সংরক্ষণ
git log                            # ইতিহাস দেখা
git log --oneline                 # সংক্ষিপ্ত ইতিহাস
git diff                          # পরিবর্তনের পার্থক্য দেখা
```

Branch & Merge

```
git branch                        # branch তালিকা
git branch feature-x              # নতুন branch তৈরি
git switch feature-x              # branch পরিবর্তন
git merge feature-x               # branch merge করা
git rebase main                   # branch rebase করা
```

Remote (GitHub এর সাথে কাজ করা)

```
git remote add origin <url>     # রিমোট যোগ করা
git push -u origin main         # লোকাল → GitHub
git pull origin main            # GitHub → লোকাল
git fetch origin                # আপডেট ডাউনলোড
```

অন্যান্য

```
git stash                       # পরিবর্তন লুকানো
git stash apply                 # stash থেকে ফেরত আনা
git cherry-pick <id>            # নির্দিষ্ট commit কপি করা
git reset filename.txt          # ফাইল unstaged করা
git rm filename.txt             # ফাইল মুছে ফেলা
git tag v1.0                    # ভার্সন ট্যাগ দেওয়া
```

12.2 দরকারি রিসোর্স

- **অফিশিয়াল ডকুমেন্টেশন**
    - Git Documentation
    - GitHub Docs
- **শেখার জন্য ওয়েবসাইট**
    - Atlassian Git Tutorials
    - Learn Git Branching (Interactive)
- **বই ও কোর্স**
    - *Pro Git* (বিনামূল্যে পড়া যাবে) → Pro Git Book
    - Udemy, Coursera, YouTube এ অসংখ্য কোর্স রয়েছে।

12.3 Glossary (বাংলায় Git & GitHub শব্দকোষ)

- **Repository (রিপোজিটরি)** → প্রজেক্টের সমস্ত কোড ও ইতিহাস যেখানে থাকে।

- **Commit (কমিট)** → একটি নির্দিষ্ট সময়ের পরিবর্তনের রেকর্ড।
- **Branch (ব্রাঞ্চ)** → মূল কোডের পাশাপাশি নতুন ফিচার বা কাজ করার আলাদা লাইন।
- **Merge (মার্জ)** → একাধিক branch কে একত্র করা।
- **Rebase (রিবেস)** → একটি branch-এর পরিবর্তন অন্য branch-এর উপরে বসানো।
- **Staging Area (স্টেজিং এরিয়া)** → ফাইল commit করার আগে যেখানে রাখা হয়।
- **HEAD** → বর্তমানে আপনি কোন commit/branch-এ আছেন সেটি নির্দেশ করে।
- **Remote** → অনলাইনে রাখা রিপোজিটরি (যেমন GitHub)।
- **Push** → লোকাল থেকে রিমোটে পরিবর্তন পাঠানো।
- **Pull** → রিমোট থেকে লোকালে পরিবর্তন আনা।
- **Fetch** → রিমোট থেকে পরিবর্তন ডাউনলোড করা কিন্তু merge না করা।
- **Fork** → অন্যের রিপোজিটরির কপি নিজের GitHub অ্যাকাউন্টে আনা।
- **Pull Request (PR)** → আপনার পরিবর্তন অন্যের প্রজেক্টে যোগ করার অনুরোধ।
- **Issues** → সমস্যা, বাগ বা নতুন ফিচারের প্রস্তাব লেখার জায়গা।
- **Release** → প্রজেক্টের একটি নির্দিষ্ট ভার্সন প্রকাশ করা।

12.4 সমাপ্তি মন্তব্য

Git এবং GitHub প্রথমে কিছুটা জটিল মনে হলেও নিয়মিত চর্চা করলে খুব সহজ মনে হবে।
এই বইয়ের উদ্দেশ্য ছিল—শিক্ষানবিশ থেকে শুরু করে মধ্যম স্তরের ব্যবহারকারীদের **বাংলায়** Git এবং GitHub শেখানো।

☞ মনে রাখবেন:

- প্রতিদিন ছোট ছোট প্র্যাকটিস করুন।
- নিজের প্রজেক্টে Git ব্যবহার শুরু করুন।
- ওপেন সোর্স প্রজেক্টে কন্ট্রিবিউট করার চেষ্টা করুন।

এভাবেই ধীরে ধীরে আপনি Git এবং GitHub-এ দক্ষ হয়ে উঠবেন। 🚀