

Git Cheat Sheet

গত ১৫ বছর কাজ করার মাঝে আমি ও আমার কর্মক্ষেত্রে সহযোগীদের গিট নিয়ে যে সকল সমস্যার সম্মুখীন হয়েছে বারংবার সেই সমস্যা ও সমাধান নিয়ে এখানে আলোচনা করেছি। আপনি যদি এই Cheat Sheet টা ভালভাবে রপ্ত করতে পারেন আমি আশা দিচ্ছি আপনার টিমে আপনি একজন মূল্যবান সম্পদ হিসাবে বিবেচ্য হবেন।

এই Git Cheat Sheet হলো আপনার জন্য একটি অল-ইন-ওয়ান গাইড — যা Git-এর মৌলিক বিষয়গুলো শিখে নিতে সহায়তা করবে।

আপনি যদি Git কমান্ড, branch ম্যানেজমেন্ট, merging, conflict resolution, অথবা version control নিয়ে দক্ষ হতে চান — এই গাইড আপনার Git journey-র জন্য আদর্শ শুরু হতে পারে।

প্রশ্নগুলো নিয়ে কাজ করার সময় কোন কিছু বুজতে সমস্যা হলে বা আরও বিস্তারিত জানতে আমাদের সাথে যোগাযোগ করুন।

লেখক পরিচিতি

আমি মোঃ সাইফুল ইসলাম, একজন ক্লাউড ইঞ্জিনিয়ার —
বর্তমানে কাজ করছি Bitstrapped-এ।
টেক ইন্ডাস্ট্রিতে দীর্ঘদিন ধরে ক্লাউড ইঞ্জিনিয়ারিং, মেশিন লার্নিং,
ব্যাকএন্ড এবং ডেটা ইঞ্জিনিয়ারিং নিয়ে কাজ করার পর,
এখন আমি নিজেকে উৎসর্গ করেছি আপনাদের মতো
শিক্ষার্থী ও প্রযুক্তিপ্রেমীদের সহায়তা করার জন্য, যাতে আপনি
শিখতে পারেন, বেড়ে উঠতে পারেন এবং প্রযুক্তির মাধ্যমে দুর্দান্ত কিছু
তৈরি করতে পারেন।



গত কয়েক বছরে, আমি কানাডাতে অসাধারণ টিমের (Google, Aramco, Boston dynamics) সঙ্গে কাজ করেছি এবং 500+ শিক্ষার্থীকে দক্ষ করে তুলেছি — যাদের অনেকেই এখন বিশ্বের শীর্ষ প্রযুক্তি প্রতিষ্ঠানে কাজ করছেন।

🌐 সাইট ও সোশ্যাল লিংক:

🔗 [Facebook](#) | 🔗 [YouTube](#) | 🔗 [LinkedIn](#) | 🔗 [Email](#)

Git এর প্রয়োজনীয় প্রশ্ন এবং উত্তর (বাংলায়)

1. আপনি কি করে অন্য origin ব্রাঞ্চ এ থাকা কোড আপনার লোকাল machine এ আনবেন?

যদি আপনি এমন একটি রিমোট ব্রাঞ্চ ব্যবহার করতে চান যা আপনার লোকাল মেশিনে এখনও নেই, তাহলে:

1. প্রথমে সমস্ত রিমোট ব্রাঞ্চ ফেচ করুন:
 - `git fetch origin`
2. তারপর নিজের ব্রাঞ্চ এ যান:
 - `git checkout` নিজের_ব্রাঞ্চের_নাম
 - `git reset --hard origin/রিমোট_ব্রাঞ্চের_নাম`
3. এবার আমরা লোকাল এর সর্বশেষ কমিট / হেডের SHA ও রিমোট_ব্রাঞ্চের SHA মিলিয়ে দেখব।
যদি একই হয় তাহলে আমাদের কাজ সফল হয়েছে
 - `git log --oneline`

2. আপনি ভুলে কোন commit আপনার লোকাল এ দিয়ে ফেললে কি করে সেটা সম্পূর্ণ ফেলে দিবে?

`git reset --hard` - পরিবর্তনগুলি বাদ দিয়ে সম্পূর্ণ ফেলে দেয়:

যদি আপনি কমিট করা সব পরিবর্তন বাদ দিতে চান এবং আগের অবস্থায় ফিরে যেতে চান

- একটি কমিট এর জন্য : `git reset --hard HEAD~1`

এই কমান্ডটি আপনার সর্বশেষ কমিটের আগের অবস্থায় ফিরে যাবে, এবং কমিটে করা সমস্ত পরিবর্তন মুছে ফেলবে। `HEAD~1` মানে "বর্তমান কমিটের আগের কমিট"। আপনি যদি একাধিক কমিট বাদ দিতে চান (যেমন সর্বশেষ ৩টি কমিট), তাহলে `HEAD~3` ব্যবহার করুন।

সতর্কতা: `git reset --hard` ব্যবহার করার সময় সাবধান থাকুন, কারণ এটি আপনার সমস্ত কমিটের পরিবর্তনও মুছে ফেলবে। আপনি `git status` দিয়ে চেক করে নিশ্চিত করতে পারেন, যে আপনার ওয়ার্কিং কপি পরিষ্কার আছে কিনা।

3. আপনি ভুলে কোন **commit** আপনার লোকাল এ দিয়ে ফেললে কি করে সেটা ফিরিয়ে আনবেন? তবে যেই ফাইল গুলো তে কাজ করেছো সেই কাজ যেন হারিয়ে না যায় ।

পদ্ধতি ১. `git reset --soft` ব্যবহার করে সর্বশেষ কমিট বাতিল করা:

এটি সবচেয়ে সহজ এবং নিরাপদ উপায়:

- `git reset --soft HEAD~1`

এই কমান্ডটি আপনার সর্বশেষ কমিট বাতিল করবে কিন্তু সেই কমিটে করা সমস্ত পরিবর্তন স্টেজিং এরিয়াতে রাখবে। এটি করার পর:

- আপনার কোড পরিবর্তন এখনও বজায় থাকবে
- পরিবর্তনগুলো স্টেজড অবস্থায় থাকবে
- আপনি চাইলে কোড আরও পরিবর্তন করতে পারেন, ফাইল আনস্টেজ করতে পারেন, বা নতুন করে কমিট করতে পারেন

পদ্ধতি ২. একাধিক কমিট ফিরিয়ে আনা:

যদি আপনি একাধিক কমিট পিছনে যেতে চান:

1. N সংখ্যক কমিট পিছনে যাওয়ার জন্য
 - `git reset --soft HEAD~N`

পদ্ধতি ৩. নির্দিষ্ট কমিটে ফিরে যাওয়া

যদি আপনি নির্দিষ্ট কোনো কমিটে ফিরে যেতে চান

1. প্রথমে কমিট হিস্ট্রি দেখুন
 - `git log --oneline`
2. নির্দিষ্ট কমিটে ফিরে যান
 - `git reset --soft <কমিট_হ্যাশ>`

4. আপনি পরপর ৩ টা **commit** ভুল করে লোকালে দিয়ে ফেললে, কি করে সেই **commit** গুলো সম্পূর্ণ ফেলে দিবে?

`git reset` ব্যবহার করা (সবচেয়ে সরল পদ্ধতি)

আপনি যদি এখনও কমিটগুলো রিমোট রিপোজিটরিতে পুশ না করে থাকেন, তাহলে `git reset` ব্যবহার করে সহজেই কমিটগুলো মুছে ফেলতে পারেন:

- `git reset --hard HEAD~3`

এটি আপনার বর্তমান কমিট থেকে পিছনে ৩টি কমিট বাদ দিয়ে আপনার লোকাল রিপোজিটরি রিসেট করবে। `--hard` অপশন ব্যবহার করার অর্থ হল আপনার ওয়ার্কিং ডিরেক্টরিও রিসেট হয়ে যাবে, অর্থাৎ কমিটগুলোতে করা সব পরিবর্তন হারিয়ে যাবে।

5. অন্য colleague এর origin ব্রাঞ্চ এ থাকা কাজ / commit গুলো আপনার লোকাল ব্রাঞ্চ এ কি করে নিয়ে আসবেন?

- [১ নং প্রশ্নের উত্তর টা দেখুন](#)

6. কি কি কারণে আপনার একটি Pull Request (PR) গিটের ভুল ব্যবহারে জনিত কারণে reject করে দিবে? কিভাবে বুঝবেন Pull Request (PR) টিতে সমস্যা আছে?

গিট ব্যবহার সংক্রান্ত কারণে Pull Request (PR) রিজেক্ট করার কারণসমূহ:

1. ভুল ব্রাঞ্চিং স্ট্র্যাটেজি: মেইন/ডেভেলপমেন্ট ব্রাঞ্চের পরিবর্তে ভুল ব্রাঞ্চে মার্জ করার চেষ্টা করা
2. কনফ্লিক্ট সমাধান না করা: মার্জ কনফ্লিক্টগুলি ঠিকভাবে সমাধান না করা
3. কমিট ইতিহাসের অসংগতি:
 - অতিরিক্ত মার্জ কমিট থাকা
 - অসংখ্য ছোট-ছোট অর্থহীন কমিট থাকা
 - কমিট বার্তাগুলি অর্থহীন না হওয়া বা প্রজেক্টের কনভেনশন অনুসরণ না করা
4. বিগ বাগ চেক: একটি PR-এ অনেক বড় পরিবর্তন করা যা রিভিউ করা কঠিন
5. রিবেসিং সমস্যা: রিবেসিং এর সময় কোড হারিয়ে যাওয়া বা ভুলভাবে রিবেস করা
6. অপ্রয়োজনীয় ফাইল:
 - বাইনারি ফাইল/বিন্ডি আর্টফ্যাক্ট কমিট করা
 - .gitignore-এ থাকা ফাইল কমিট করা
 - IDE কনফিগারেশন ফাইল অনাবশ্যকভাবে যোগ করা

কিভাবে Pull Request (PR)-এ সমস্যা চিহ্নিত করবেন:

1. PR ডিফারেন্স চেক করুন:
 - যে ফাইলগুলো পরিবর্তন হয়েছে সেগুলো প্রয়োজনীয় কিনা
 - পরিবর্তনের পরিমাণ যুক্তিসঙ্গত কিনা দেখুন

2. কমিট হিস্টরি পরীক্ষা করুন:

- `git log` ব্যবহার করে কমিট বার্তাগুলো পর্যালোচনা করুন
- `git blame` ব্যবহার করে কে কোন পরিবর্তন করেছে তা দেখুন

3. CI/CD পাইপলাইন রেজাল্ট দেখুন:

- অটোমেটেড টেস্ট ফেল করে কিনা
- লিন্টিং/কোড স্টাইল চেক পাস করে কিনা

4. কোড রিভিউ টুল ব্যবহার করুন:

- GitHub/GitLab এর বিল্ট-ইন রিভিউ টুল
- SonarQube বা অন্যান্য স্ট্যাটিক কোড অ্যানালাইসিস টুল

5. ব্রাঞ্চ কমপেয়ার করুন:

- `git diff main...feature-branch --stat`

6. PR টেমপ্লেট নিয়ম মেনে চলেছে কিনা দেখুন:

- প্রয়োজনীয় ডকুমেন্টেশন আছে কিনা
- PR সম্পর্কিত টাস্ক/ইস্যু রেফারেন্স করা আছে কিনা

7. মার্জ কনফ্লিক্ট চেক করুন:

- `git checkout main`
- `git pull`
- `git checkout feature-branch`
- `git merge main`

সফল PR-এর জন্য সর্বোত্তম অনুশীলন হলো নিয়মিত ছোট PR জমা দেওয়া, ভাল কমিট বার্তা লেখা, এবং PR জমা দেওয়ার আগে নিজেই একবার রিভিউ করা।

7. **Origin** ব্রাঞ্চে ভুল করে ২ টি **commit** দিয়েছেন। সেই **commit** গুলো এখন অন্যরা তাদের লোকালে নিয়ে কাজ করছে। কি করে সেই ভুল **commit** গুলো ফিরিয়ে আনবেন ও অন্যদের জানাবেন?

কমিট ফিরিয়ে আনার পদ্ধতি

ধরি আমি `my_work` ব্রাঞ্চ এর `origin (origin my_work)` এ পুশ করে ফেলেছি ভুলে

পদ্ধতি ১. `git revert` ব্যবহার করুন (সবচেয়ে নিরাপদ পদ্ধতি)

যেহেতু কমিটগুলো ইতিমধ্যে শেয়ার করা হয়েছে, তাই `git revert` ব্যবহার করা সবচেয়ে নিরাপদ এবং সুপারিশকৃত উপায়:

1. প্রথমে আপডেটেড ব্রাঞ্চে যান:

- `git checkout my_work`

2. কমিট হিস্ট্রি দেখুন:

- `git log --oneline`

3. ভুল কমিটগুলো রিভার্ট করুন (প্রতিটি কমিটের জন্য আলাদা রিভার্ট):

1. `git revert <ভুল_কমিট_হ্যাশ_১>`
2. `git revert <ভুল_কমিট_হ্যাশ_২>`

4. আপনি nano বা vi এডিটর এ আপনার একটি নতুন কমিট মেসেজ পাবেন ।
5. নিজের মত করে সেটা পরিবর্তন করুন ও কমিট করুন ।
6. এবার git log দিলে দেখবেন নতুন কমিটটি আপনি পেয়েছেন সবার উপড়ে ।
7. রিভার্ট কমিটগুলো পুশ করুন
 - `git push origin my_work`

`git revert` ব্যবহার করাই সবচেয়ে নিরাপদ পদ্ধতি। কারণ এই পদ্ধতিতে কমিটের ইতিহাস পরিবর্তন না করে, সঠিক ভাবে সমস্যার সমাধান করা হয় - যা সবচেয়ে গুরুত্বপূর্ণ।

পদ্ধতি ২. `git reset` এবং ফোর্স পুশ (শুধুমাত্র যদি অন্য কেউ এখনো পুশ না করে থাকে)

সতর্কতা: এই পদ্ধতি শুধুমাত্র তখনই ব্যবহার করুন যখন আপনি নিশ্চিত যে অন্য কেউ এই কমিট এখনো পুশ করেনি।

1. প্রথমে আপডেটেড ব্রাঞ্চে যান
 - `git checkout my_work`
2. দুই কমিট আগে ফিরে যান
 - `git reset --soft HEAD~2`
3. অথবা নির্দিষ্ট কমিটে ফিরে যান
 - `git reset --soft <সঠিক_কমিট_হ্যাশ>`
4. ফোর্স পুশ করুন (সতর্কতার সাথে)
 - `git push --force origin <ব্রাঞ্চ_নাম>`

এটি আপনার লোকাল ও রিমোট কমিটের ইতিহাস পরিবর্তন করে, তাই সতর্কতার সাথে ব্যবহার করুন।

টিম সদস্যদের জন্য নির্দেশাবলী:

1. `git fetch origin` বা `git pull` দিয়ে নতুন পরিবর্তন নিন
2. লোকাল ব্রাঞ্চে থাকা কাজ স্ট্যাশ করুন: `git stash`
3. মেইন ব্রাঞ্চ রিসেট করুন: `git checkout main && git reset --hard origin/main`
4. সঠিক ফিচার ব্রাঞ্চে যান: `git checkout` সঠিক-ফিচার-ব্রাঞ্চ
5. স্ট্যাশ করা কাজ যদি প্রয়োজন হয় সেক্ষেত্রে অ্যাপ্লাই করুন: `git stash pop`

8. সর্বশেষ 4 টি **commit** কে আপনি 1 টি **commit** আনতে চান । কি করবেন ? ধরে নেন **commit** গুলো **origin** এ পুশ করা হয়নি এখনো ।

পদ্ধতি ১: সহজ রিসেট ও কমিট পদ্ধতি

সর্বশেষ ৪টি কমিট বাদ দিয়ে **HEAD** কে সেট করুন, কিন্তু ফাইলের পরিবর্তন রাখুন

- `git reset --soft HEAD~4`

এবার একটি নতুন কমিট করুন

- `git commit -m "Comprehensive commit message describing all changes"`

সফট রিসেট শুধু HEAD-কে সেই সর্বশেষ কমিটে পয়েন্ট করে যা আপনি স্কোয়াশ করতে চান না। না ইনডেক্স, না ওয়ার্কিং ট্রি সফট রিসেট দ্বারা প্রভাবিত হয়।

বিশেষ দৃষ্টব্য

যদি স্কোয়াশ করার সময় কোন কনফ্লিক্ট দেখা দেয়, Git আপনাকে সেগুলি সমাধান করতে বলবে। এক্ষেত্রে ফাইলগুলিতে কনফ্লিক্ট মার্কার (`<<<<<<, =====, >>>>>>`) দেখা যাবে। এগুলি সমাধান করুন, তারপর `git add` দিয়ে সমাধানকৃত ফাইলগুলি স্টেজ করুন এবং `git rebase --continue` দিয়ে রিবেস প্রক্রিয়া চালিয়ে যান।

মনে রাখবেন, শুধুমাত্র লোকাল কমিটগুলির জন্যই এই পদ্ধতিগুলি নিরাপদ। যদি কমিটগুলি ইতিমধ্যে পুশ করা হয়ে থাকে, তাহলে ইতিহাস পুনর্লিখন করা এড়ানো উচিত বা বিশেষ সতর্কতা অবলম্বন করা দরকার।

9. ধরে নেন **commit** গুলো **origin** এ পুশ করা হয়েছে। এখন আপনি ১ টি **commit** কে আনতে চান। কি ভাবে আনবেন?

পদ্ধতি ১. `git revert` দিয়ে কমিট উল্টানো (সবচেয়ে নিরাপদ উপায়)

এই পদ্ধতিতে আপনি কমিটটিকে বাতিল না করে তার পরিবর্তনগুলোকে উল্টিয়ে দেন:

1. কমিট আইডি খুঁজে বের করুন
 - `git log --oneline`
2. কমিটটি revert করুন
 - `git revert <বাতিল_করতে_চাওয়া_কমিটের_হ্যাশ>`
3. আপনি nano বা vi এডিটর এ আপনার একটি নতুন কমিট মেসেজ পাবেন।
4. নিজের মত করে সেটা পরিবর্তন করুন ও কমিট করুন।
5. এবার `git log` দিলে দেখবেন নতুন কমিটটি আপনি পেয়েছেন সবার উপড়ে।
6. পরিবর্তনগুলো পুশ করুন
 - `git push origin <ব্রাঞ্চ_নাম>`

এটি সবচেয়ে নিরাপদ উপায় কারণ এটি মূল কমিটের ইতিহাস পরিবর্তন করে না, বরং একটি নতুন কমিট তৈরি করে যা আগের পরিবর্তনগুলোকে উল্টায়।

পদ্ধতি ২. `git reset` এবং ফোর্স পুশ (শুধুমাত্র যদি অন্য কেউ এখনো পুল না করে থাকে)

সতর্কতা: এই পদ্ধতি শুধুমাত্র তখনই ব্যবহার করুন যখন আপনি নিশ্চিত যে অন্য কেউ এই কমিট এখনো পুল করেনি।

5. এক কমিট আগে ফিরে যান
 - `git reset --soft HEAD~1`
6. অথবা নির্দিষ্ট কমিটে ফিরে যান
 - `git reset --soft <সঠিক_কমিট_হ্যাশ>`
7. ফোর্স পুশ করুন (সতর্কতার সাথে)
 - `git push --force origin <ব্রাঞ্চ_নাম>`

এটি আপনার লোকাল ও রিমোট কমিটের ইতিহাস পরিবর্তন করে, তাই সতর্কতার সাথে ব্যবহার করুন।

যে সতর্কতা অবলম্বন করবেন:

1. **মাস্টার/মেইন ব্রাঞ্চে এড়িয়ে চলুন:** শেয়ারড মাস্টার/মেইন ব্রাঞ্চে এই ধরনের পরিবর্তন এড়িয়ে চলাই উত্তম।
2. **টিমকে জানান:** আপনি ফোর্স পুশ করার আগে টিম মেম্বারদের জানান যাতে তারা আপ-টু-ডেট থাকে।
3. **নিশ্চিত হোন:** ফোর্স পুশ করার আগে, নিশ্চিত হোন যে আপনি সঠিক কমিট বাতিল করছেন।
4. **ব্রাঞ্চ প্রোটেকশন:** যদি ব্রাঞ্চে প্রোটেকশন সেট করা থাকে, তাহলে ফোর্স পুশ সম্ভব নাও হতে পারে।

টিম এনভায়রনমেন্টে সবসময় `git revert` ব্যবহার করাই সবচেয়ে নিরাপদ, কারণ এটি শেয়ারড কমিটের ইতিহাস পরিবর্তন করে না।

10. একটি ফাইল যা আসলে গিট এ থাকার কথা না কিন্তু ভুলে পুশ করে ফেলেছেন। কি করে সেটাকে আর যেন কেও **origin** পুশ না করে সেটা নিশ্চিত করবেন ?

1. প্রথমে `.gitignore` ফাইলে সেই ফাইলটি যোগ করুন:
 - `echo "ফাইলের_নাম" >> .gitignore`
2. তারপর গিট রেপোজিটরি থেকে ফাইলটি সরাতে হবে, কিন্তু লোকাল সিস্টেম থেকে নয়। এর জন্য নিচের কমান্ড ব্যবহার করুন:
 - `git rm --cached ফাইলের_নাম`
3. এবার নতুন কমিট করুন:
 - `git commit -m "Remove ফাইলের_নাম from git tracking"`
4. পরিবর্তন origin-এ পুশ করুন:
 - `git push origin আপনার_ব্রাঞ্চ`

এই পদ্ধতিতে:

- ফাইলটি আপনার লোকাল সিস্টেমে থাকবে
- গিট আর ফাইলটি ট্র্যাক করবে না
- `.gitignore` ফাইলে ফাইলটি যোগ করার ফলে টিমের অন্য সদস্যরাও এটিকে পুশ করতে পারবে না

যদি ফোল্ডার অথবা একাধিক একই ধরনের ফাইল থাকে, তাহলে `.gitignore`-এ প্যাটার্ন ব্যবহার করতে পারেন:

1. একটি ফাইল ইগনোর করতে
 - `sensitive_config.json`
2. একটি ফোল্ডার ইগনোর করতে
 - `logs/`
3. একই ধরনের সকল ফাইল ইগনোর করতে
 - `*.log`
 - `*.tmp`

সবাইকে এই পরিবর্তন সম্পর্কে জানাতে ভুলবেন না, যাতে তারা লোকালি `.gitignore` আপডেট করে নেয়।

11. আপনি একটি ফোল্ডার কে গিটের বাইরে রাখতে চান। কি করে রাখবেন ?

একটি ফোল্ডারকে Git-এর বাইরে রাখার পদ্ধতি

Git প্রজেক্টের মধ্যে কিছু ফোল্ডার বা ফাইল রাখতে চাইলে যেগুলো ট্র্যাক করা উচিত নয়, সেগুলো Git-এর বাইরে রাখার জন্য নিম্নলিখিত পদ্ধতিগুলো ব্যবহার করতে পারেন:

১. `.gitignore` ফাইল ব্যবহার করা

এটি সবচেয়ে সাধারণ এবং প্রস্তাবিত পদ্ধতি:

কীভাবে `.gitignore` ফাইল তৈরি করবেন:

1. আপনার প্রজেক্টের রুট ডিরেক্টরিতে `.gitignore` নামে একটি ফাইল তৈরি করুন:
 - `touch .gitignore`
2. যে ফোল্ডারকে আপনি গিটের বাইরে রাখতে চান তার নাম বা পথ `.gitignore` ফাইলে যোগ করুন:
 - সম্পূর্ণ ফোল্ডার বাদ দিতে
 - ❖ `node_modules/`
 - ❖ `build/`
 - ❖ `temp_folder/`
3. নির্দিষ্ট পথে থাকা ফোল্ডার বাদ দিতে
 - `path/to/specific/folder/`
4. উদাহরণস্বরূপ, যদি আপনি একটি `private_data` নামক ফোল্ডার Git থেকে বাদ দিতে চান:
 - `private_data/`

সুবিধা:

- প্রজেক্টের সবার জন্য একই নিয়ম প্রযোজ্য হয়
- রিপোজিটরিতে প্রযোজ্য নিয়মগুলো রেকর্ড করা থাকে
- অন্যান্য ডেভেলপাররা একই নিয়ম অনুসরণ করবে

২. ইতিমধ্যে ট্র্যাক করা হয়ে গেছে এমন ফোল্ডার বাদ দেওয়া

যদি ফোল্ডারটি ইতিমধ্যে Git এ ট্র্যাক করা হয়ে থাকে, তাহলে:

1. ফোল্ডারটিকে Git ট্র্যাকিং থেকে সরিয়ে ফেলুন (ফাইলগুলো হারিয়ে যাবে না)
 - `git rm -r --cached folder_name`
2. আপনার .gitignore ফাইলে ফোল্ডারটি যোগ করুন
 - `echo "folder_name/" >> .gitignore`
3. এখন পরিবর্তনগুলো কমিট করুন
 - `git add .gitignore`
 - `git commit -m "Remove folder_name from git tracking"`

সাবধানতা

- .gitignore ফাইলে পথ যোগ করলে শুধুমাত্র ভবিষ্যতে ট্র্যাক না করা ফাইলগুলো প্রভাবিত হবে
- ইতিমধ্যে ট্র্যাক করা হয়ে থাকা ফাইল/ফোল্ডার অবশ্যই `git rm --cached` দিয়ে আনট্র্যাক করতে হবে
- সেনসিটিভ ডেটা কখনো Git রিপোজিটরিতে রাখবেন না, এমনকি .gitignore এর মাধ্যমে বাদ দিলেও, কারণ পূর্বের কমিট হিস্টরিতে সেটি থাকতে পারে

সারাংশ

একটি ফোল্ডারকে গিট থেকে বাদ দেওয়ার সবচেয়ে সাধারণ এবং প্রস্তাবিত উপায় হলো প্রজেক্টের রুট ডিরেক্টরিতে একটি .gitignore ফাইল তৈরি করে তাতে ফোল্ডারের পথ যোগ করা।

12. লোকালের একটি ব্রাঞ্চ যে origin এর একই ব্রাঞ্চের সাথে আপডেট আছে, মানে origin এর ব্রাঞ্চে যা যা কাজ করা হইসে তা আপনার লোকালের ব্রাঞ্চে আছে কি করে চেক করবেন ?

ধরুন আপনি my_branch এ এখন আছেন। আপনি জানেন না my_branch এর origin এ কেও কিছু পুশ করেছে কিনা? মানে origin এ থাকা ব্রাঞ্চ হতে লোকাল এ থাকা ব্রাঞ্চ কতটুকু আগিয়ে বা পিছিয়ে আছে কিনা দেখবেন। সেটি কি করে করবেন তার ধারণা দেওয়া হচ্ছে।

git status ব্যবহার করা

প্রথমে আপনি git fetch করে নিবেন। এতে origin এ থাকা সব কিছু আপনার লোকাল এ চলে আসবে।

- `git fetch`

এবার `git status` কমান্ড ব্যবহার করা

- `git status`

সাধারণ `git status` কমান্ড দেখাবে আপনার বর্তমান ব্রাঞ্চ রিমোটের তুলনায় কতগুলি কমিট আগে/পিছে আছে। নিশ্চিত করুন যে সমস্ত রিমোট সঠিকভাবে কনফিগার করা আছে।

এই কমান্ডের আউটপুট নিম্নলিখিত বার্তাগুলির মধ্যে একটি দেখাবে:

- **যদি সমান হয়:** Your branch is up to date with 'origin/branch_name'.
- **যদি আপনার কমিট বেশি থাকে:** Your branch is ahead of 'origin/branch_name' by X commits.
- **যদি origin-এ কমিট বেশি থাকে:** Your branch is behind 'origin/branch_name' by X commits.
- **যদি উভয়ই পরিবর্তিত হয়ে থাকে:** Your branch and 'origin/branch_name' have diverged.

13. আপনি কাজ করছেন। হঠাৎ আপনার একজন **colleague** এর ব্রাঞ্চে গিয়ে দেখা লাগবে তার কোড আপনার লোকালে ভালোভাবে চলে কিনা। আপনার গিটে কিছু **untrack** ফাইল আছে। এখন কি করবেন?

git stash ব্যবহার করা (সবচেয়ে উপযুক্ত পদ্ধতি)

`git stash` কমান্ড আপনার অসম্পূর্ণ কাজকে সাময়িকভাবে সেভ করে রাখবে, যাতে আপনি অন্য ব্রাঞ্চে যেতে পারেন এবং পরে আবার আপনার কাজে ফিরে আসতে পারেন:

1. আনট্র্যাকড ফাইলসহ সব পরিবর্তন স্ট্যাশ করুন
 - `git stash -u`
2. কলিগের ব্রাঞ্চে যান
 - `git checkout কলিগের_ব্রাঞ্চ_নাম`
3. যদি আপনার কলিগের_ব্রাঞ্চ_নাম এর `origin` এর সাথে uptodate হওয়া লাগে তবে [লিঙ্ক](#) দেখুন
4. কলিগের কোড চেক/টেস্ট করুন
5. আপনার মূল ব্রাঞ্চে ফিরে আসুন
 - `git checkout আপনার_মূল_ব্রাঞ্চ`
6. আপনার স্ট্যাশ করা পরিবর্তন ফিরিয়ে আনুন
 - `git stash pop`

`git stash` ব্যবহার করে আপনি কমিট না করেই আপনার পরিবর্তনগুলি লোকালি সেভ করতে পারেন, ব্রাঞ্চ পরিবর্তন করতে পারেন, অন্যান্য অপারেশন সম্পাদন করতে পারেন, এবং তারপরে প্রয়োজন হলে লোকালি স্ট্যাশ করা পরিবর্তনগুলি পুনরায় প্রয়োগ করতে পারেন।

বিশেষ দ্রষ্টব্য:

- `-u` ক্যাগ ব্যবহার করা গুরুত্বপূর্ণ, এটি আনট্র্যাকড ফাইলগুলিও স্ট্যাশে অন্তর্ভুক্ত করে
- যদি আপনি ইগনোরড ফাইলগুলিও স্ট্যাশ করতে চান, তাহলে `-a` ক্যাগ ব্যবহার করুন

14. Track, untrack ও Staged এর মধ্যে পার্থক্য কি ?

Tracked ফাইল

Tracked ফাইল হল সেইসব ফাইল যা Git ভার্সন কন্ট্রলের অধীনে আছে। আপনি যদি একটি ফাইল `git add` বা `git commit` করে থাকেন, তাহলে Git সেই ফাইলকে ট্র্যাক করে।

বৈশিষ্ট্য:

- Git এদের সম্পর্কে জানে এবং স্বয়ংক্রিয়ভাবে এদের অবস্থা (unmodified, modified বা staged) নিরীক্ষণ করে
- আপনি এদের পরিবর্তনের ইতিহাস দেখতে পারেন
- আপনি এদের আগের সংস্করণে ফিরে যেতে পারেন

উদাহরণ: যদি আপনি `file.txt` নামে একটি ফাইল তৈরি করে `git add file.txt` কমান্ড ব্যবহার করেন, এবং তারপর `git commit` করেন, তাহলে `file.txt` একটি tracked ফাইল হবে।

Untracked ফাইল

Untracked ফাইল হল সেইসব ফাইল যা আপনার প্রজেক্টে আছে কিন্তু Git বর্তমানে ট্র্যাক করে না। এর অর্থ হল Git এই ফাইলগুলির পরিবর্তনের ইতিহাস রাখবে না।

বৈশিষ্ট্য:

- Git এদের সম্পর্কে কিছুই জানে না এবং এদের পরিবর্তন ট্র্যাক করে না
- `git status` কমান্ড ব্যবহার করলে, এগুলি "Untracked files" বিভাগে দেখাবে
- এগুলি কখনও কমিট করা হয়নি বা স্টেজিং এরিয়ায় যোগ করা হয়নি

উদাহরণ: যদি আপনি আপনার প্রজেক্টে একটি নতুন ফাইল `newfile.txt` তৈরি করেন কিন্তু এখনও `git add` বা `git commit` না করে থাকেন, তাহলে এটি একটি untracked ফাইল।

Staged ফাইল


Staged হল tracked ফাইলের একটি "উপ-অবস্থা"। ফাইলগুলি tracked বা untracked হতে পারে। যদি তারা tracked হয় তবে তারা unmodified, modified বা staged হতে পারে।

বৈশিষ্ট্য:

- এগুলি ট্র্যাকড ফাইল যা পরবর্তী কমিটের জন্য প্রস্তুত করা হয়েছে
- Git এদের বর্তমান অবস্থা স্টেজিং এরিয়ায় সংরক্ষণ করে (আপনার পরবর্তী কমিটে অন্তর্ভুক্ত করতে)
- git status কমান্ড ব্যবহার করলে, এগুলি "Changes to be committed" বিভাগে দেখাবে

উদাহরণ: যদি আপনি file.txt পরিবর্তন করেন এবং তারপর git add file.txt ব্যবহার করেন, তাহলে এটি একটি staged ফাইল হবে।

এই Cheat Sheet ভাল করে পড়তে বা নিজে হাতে প্রয়োগ করতে কোন সমস্যার সম্মুখীন হলে আমাদের সাথে যোগাযোগ করতে পারেন। আপনার যদি কোনও প্রশ্ন, পরামর্শ, বা কোর্স সম্পর্কে জানতে ইচ্ছা থাকে — আমাদের সঙ্গে যোগাযোগ করুন নিচের দেওয়া নাম্বারে।

 যোগাযোগ করুন: **01676743076**