Rebasing - - -

**Rebasing को सरल बुझाई (नेपालीमा)**

Git मा, दुई मुख्य तरिका हुन्छन् कुनै पनि शाखाका परिवर्तनलाई अर्को शाखामा मिलाउन: **merge (मर्ज)** र **rebase (रिबेस)**। यहाँ Rebase के हो, कसरी प्रयोग गर्ने, किन यो उपयोगी छ, र कहिले यसलाई प्रयोग नगर्नु राम्रो हुन्छ भन्ने कुरा बुझ्न सकिन्छ।

**Rebase के हो?**

Rebase भनेको एउटै शाखाका commits (परिवर्तन) लाई अर्को शाखाको टुप्पो (head) मा **पुनः लागू** (re-apply) गर्ने प्रक्रिया हो। यसले branch (शाखा) को इतिहासलाई सफा र linear (सीधा र व्यवस्थित) बनाउँछ।

A diagram of a flowchart

Description automatically generated

**उदाहरण:**

तपाईंले एउटै common base commit (C2) बाट शाखा विभाजन गर्नुभयो र दुई छुट्टा-छुट्टै commits (C3 र C4) बनाउनुभयो।

C3 (शाखा A)

/

C2 ---

\

C4 (शाखा B)

**Merge (पारम्परिक तरीका)**

Merge गर्दा, Git ले तीन-वटा commits लाई हेर्छ:

1. **दुई शाखाका टुप्पो commits** (C3 र C4)।
2. **सबैभन्दा पछिल्लो साझा आधार** commit (C2)।

Merge ले नयाँ snapshot (M) र commit सिर्जना गर्दछ जसले दुई शाखाहरूलाई समायोजन गर्छ।

A diagram of a work history

Description automatically generated

**Merge को परिणाम:**

C3

/ \

C2 --- M (merge commit)

\ /

C4

**Rebase (वैकल्पिक तरीका)**

Rebase ले merge commit बनाउँदैन, बरु commit हरूलाई अर्को शाखाको टुप्पोमा **पुनः लागू** गर्दछ।

**Rebase के गर्छ?**

* Rebase गर्दा, Git ले C3 commit लाई C4 को टुप्पोमा राख्छ।
* इतिहास linear (सीधा) र सफा देखिन्छ।

**Rebase को परिणाम:**

C2 --- C4 --- C3

**Rebase प्रयोग गर्ने बेला**

1. **Linear इतिहास चाहिएको बेला**:
   * तपाईंलाई project को इतिहास सफा र सरल देखिनुपर्छ भने।
2. **Collaboration (सहकार्य) अघि**:
   * शाखालाई shared गर्ने (push) अघि Rebase गर्दा इतिहास organized देखिन्छ।

**Rebase प्रयोग नगर्नु पर्ने बेला**

1. **पहिले नै public branch मा commits भएमा**:
   * यदि commits पहिले नै साझा गरिएका छन् भने, Rebase गर्दा अन्य प्रयोगकर्ताहरूको काम प्रभावित हुन सक्छ।
2. **Collaborative branches मा काम गर्दा**:
   * Merge नै सुरक्षित विकल्प हो।

**निष्कर्ष**

Rebase र Merge दुवै परिवर्तनलाई समायोजन गर्ने तरिका हुन्। तर, Rebase project को इतिहासलाई सफा बनाउन सहयोगी हुन्छ, तर यसलाई सावधानीपूर्वक प्रयोग गर्नु पर्छ।

**Rebasing को प्रक्रियालाई बुझ्ने (नेपालीमा)**

**Rebase** भनेको शाखाको परिवर्तन (commits) लाई अर्को शाखाको टुप्पो (head) मा पुनः लागू (reapply) गर्ने प्रक्रिया हो। यो प्रक्रियाले शाखा को इतिहासलाई linear (सफा र सीधा) बनाउँछ।

Rebasing प्रक्रिया - - -

**स्थिति:**

तपाईंले दुई शाखा (master र experiment) बनाएर काम गरिरहनुभएको छ। experiment शाखामा नयाँ commit (C4) थपिएको छ। अब, तपाईं यो परिवर्तनलाई master शाखामा integrate गर्न चाहनुहुन्छ।

**शाखाको प्रारम्भिक स्थिति:**

C4 (experiment)

/

C3 ---

\

(master)

**1. Rebase आदेश चलाउने**

Rebasing गर्दा, तपाईं experiment शाखामा जानुहुन्छ र त्यसलाई master शाखामा rebasing गर्नुहुन्छ।

**Command:**

$ git checkout experiment

$ git rebase master

**Output:**

First, rewinding head to replay your work on top of it...

Applying: added staged command

**2. Rebasing प्रक्रिया**

Rebasing गर्दा निम्न चरणहरू हुन्छन्:

1. **साझा आधार (common ancestor)** commit (C3) लाई फेला पारिन्छ।
2. experiment शाखामा भएका commits (C4) लाई **diffs** का रूपमा temporary फाइलमा सुरक्षित गरिन्छ।
3. experiment शाखाको टुप्पोलाई master शाखाको टुप्पोमा **reset** गरिन्छ।
4. सुरक्षित गरिएका diffs क्रमैसँग लागू (apply) गरिन्छ।

**Rebase को परिणाम:**

C3 --- C4 (experiment, अब master को टुप्पोमा छ)

**3. Fast-Forward Merge**

अब master शाखामा गएर experiment शाखालाई merge गर्न सकिन्छ। किनकि experiment शाखा अब master शाखाको टुप्पोमा linear रूपमा छ, Git fast-forward merge गर्छ।

**Command:**

$ git checkout master

$ git merge experiment

**Final स्थिति:**

C3 --- C4 (master र experiment दुवै)

A diagram of a branch

Description automatically generated

**Rebasing को लाभ**

1. **सफा इतिहास:** Rebase गर्दा branch को इतिहास linear र स्पष्ट देखिन्छ।
2. **Collaborative प्रोजेक्टमा सजिलो:** शाखा साझा (push) गर्नु अघि Rebase गर्दा conflict कम हुने सम्भावना हुन्छ।

**ध्यान दिनुपर्ने कुरा**

1. **Rebase गरेर साझा नगर्नु:** यदि commits पहिले नै साझा गरिएका छन् भने Rebase गर्दा अन्य प्रयोगकर्ताहरूको काममा समस्या आउन सक्छ।
2. **Conflict को सम्भावना:** Rebase गर्दा conflicts उत्पन्न भएमा तिनलाई समाधान गर्नुपर्छ।

Rebase सही तरिकाले प्रयोग गर्दा branch को इतिहासलाई सरल र व्यवस्थित बनाउन सकिन्छ।

## Rebasing vs Merging: फरक बुझ्नुस् (सरल व्याख्या)

**अन्तिम नतिजा उस्तै हुन्छ**

**Merge** र **Rebase** प्रयोग गरेर परिवर्तनहरू एकीकृत गर्दा अन्तिम **Snapshot (अन्तिम कोड)** उस्तै हुन्छ। फरक केवल **इतिहास कसरी रेकर्ड गरिन्छ** भन्नेमा हुन्छ।

**Rebasing: सफा रेखीय इतिहासको लागि**

1. **रेखीय इतिहास (Linear History):**  
   Rebasing गर्दा शाखा (branch) को इतिहास यस्तो देखिन्छ कि सबै काम अनुक्रमिक रूपमा (एक लाइनमा) भएको छ, चाहे त्यो वास्तवमा समानान्तर (parallel) मा भएको थियो।  
   यसले परियोजनाको इतिहासलाई सफा र बुझ्न सजिलो बनाउँछ।

**Rebased History:**

C1 --- C2 --- C3 --- C4' (रेखीय इतिहास)

1. **योगदानको लागि उपयुक्त (Ideal for Contributions):**  
   जब तपाईं कुनै यस्तो प्रोजेक्टमा योगदान दिनुहुन्छ जुन तपाईंले मर्मत (maintain) गर्नुहुन्न, Rebasing उपयोगी हुन्छ।
   * तपाईंले आफ्नो कामलाई origin/master (मुख्य शाखा) मा **rebase** गर्न सक्नुहुन्छ।
   * यसले तपाईंको commits सफा रूपमा लागू गर्न सुनिश्चित गर्दछ।
   * परियोजना मर्मतकर्ताले (maintainer) केवल **fast-forward** वा सफा **apply** गरेर तपाईंको कामलाई समाहित गर्न सक्छ।

**उदाहरण वर्कफ्लो (Example Workflow):**

$ git fetch origin

$ git rebase origin/master

**Merge: समानान्तर इतिहासको लागि**

जब तपाईं merge गर्नुहुन्छ, Git ले तपाईंको शाखाको इतिहास र तपाईंले merge गर्न खोजेको शाखाको इतिहास दुवै रेकर्ड गर्छ र तिनीहरूलाई नयाँ commit (merge commit) मा मिलाउँछ।

**Merged History:**

C1 --- C2 --- C3 --- C5 (Merge Commit)

\ /

C4

**Rebase (C4')** र **Merge (C5)** को अन्तिम Snapshot एउटै भए पनि, merge को इतिहासले समानान्तर काम र एकीकृत हुने बिन्दु (integration point) देखाउँछ।

**Rebasing कहिले गर्ने?**

1. जब तपाईं आफ्नो commit history सफा बनाउन चाहनुहुन्छ (जस्तै, push गर्नु अघि)।
2. जब तपाईं चाहनुहुन्छ कि तपाईंको परिवर्तनहरू (changes) मुख्य शाखाका पछिल्ला commits पछि अनुक्रमिक रूपमा लागू भएको देखियोस्।
3. जब तपाईं कुनै प्रोजेक्टमा योगदान गर्दै हुनुहुन्छ जुन तपाईंले मर्मत गर्नुहुन्न, ताकि मर्मतकर्ताले merge गर्दा झन्झटको सामना गर्न नपरोस्।

अधिक चाखलाग्दो Rebases - - -  
तपाईं आफ्नो rebase replay कुनै अर्को शाखामा पनि गर्न सक्नुहुन्छ, जुन rebase target शाखा होइन। एउटा यस्तो इतिहासलाई लिनुहोस्, जस्तै अर्को topic branch बाट छुट्टिएको कुनै topic branch को इतिहास। उदाहरणका लागि, तपाईंले server शाखा (topic branch) बनाएर आफ्नो प्रोजेक्टमा केही server-side functionality थप्नुभयो र एउटा commit गर्नुभयो। त्यसपछि, तपाईंले client शाखा (client-side changes) को लागि अर्को शाखा बनाएर केही पटक commit गर्नुभयो। अन्ततः, तपाईं आफ्नो server शाखामा फर्किनुभयो र त्यहाँ थप commits गर्नुभयो।

यो कार्यशैलीले तपाईंलाई तपाईंको विकास प्रक्रिया अझ लचिलो र प्रभावकारी बनाउने मौका दिन्छ।

A diagram of a tree

Description automatically generated

मानौँ, तपाईंले आफ्नो client-side परिवर्तनहरू मुख्य शाखामा (mainline) release को लागि merge गर्ने निर्णय गर्नुभयो, तर server-side परिवर्तनहरूलाई अझ परीक्षण नगर्दासम्म रोकिराख्न चाहनुहुन्छ। तपाईं client शाखामा भएका परिवर्तनहरू (C8 र C9) जसले server शाखामा केही असर गर्दैनन्, ती परिवर्तनहरू master शाखामा replay गर्न सक्नुहुन्छ --onto विकल्पको प्रयोग गरेर:

$ git rebase --onto master server client

यसको अर्थ के हो भने, "client शाखा लिनुहोस्, यो server शाखाबाट छुटिएको समयपछि भएका सबै परिवर्तन (patches) पत्ता लगाउनुहोस्, र ती परिवर्तनहरू client शाखामा replay गर्नुहोस्, यसरी लाग्छ कि ती परिवर्तनहरू सीधा master शाखाबाट सुरु भएको हो।"

यो प्रक्रिया अलि जटिल छ, तर परिणाम निकै चाखलाग्दो र उपयोगी हुन्छ। यो तरिकाले तपाईं आफ्नो client-side परिवर्तनहरूलाई mainline मा merge गर्न सक्नुहुन्छ बिना server-side परिवर्तनहरूलाई असर पारेर।

A diagram of a flowchart

Description automatically generated

अब तपाईं आफ्नो master शाखालाई fast-forward गर्न सक्नुहुन्छ, जसले client शाखामा rebased परिवर्तनहरूलाई master मा merge गर्दछ।

$ git checkout master

$ git merge client

A diagram of a tree

Description automatically generated

मानौँ तपाईंले आफ्नो **server** शाखालाई पनि **merge** गर्ने निर्णय गर्नुभयो। तपाईं git rebase <basebranch> <topicbranch> प्रयोग गरेर **server** शाखालाई **master** शाखामा **rebase** गर्न सक्नुहुन्छ। यसले **server** शाखालाई **checkout** नगरी नै **master** शाखामा **replay** गर्छ:

$ git rebase master server

यसले server शाखामा भएका परिवर्तनहरूलाई master शाखाका परिवर्तनहरूको माथि replay गर्छ।

A diagram of a tree

Description automatically generated with medium confidence

त्यसपछि, तपाईं base शाखा (master) लाई fast-forward गर्न सक्नुहुन्छ:

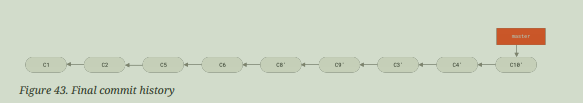
$ git checkout master

$ git merge server

अब, किनभने सारा काम integrate भइसकेको छ र ती शाखाहरू आवश्यक छैनन्, तपाईं client र server शाखाहरू हटाउन सक्नुहुन्छ:

$ git branch -d client

$ git branch -d server



The perils of rebasing

Rebasing शक्तिशाली हुन सक्छ, तर यससँग केही जोखिमहरू पनि छन्, विशेष गरी यसलाई गलत तरिकामा प्रयोग गर्दा। यहाँ तपाईंलाई सम्झाउनुपर्ने मुख्य कुरा हो:

**तपाईंको रिपोजिटरी बाहिर भएका र अरुले काम आधार बनाएको commits लाई rebase नगर्नुहोस्।**

यदि तपाईं यो guideline अनुसरण गर्नुहुन्न भने, त्यसले अराजकता ल्याउन सक्छ। तपाईं इतिहासलाई पुनः लेख्दै हुनुहुन्छ, र अरुले तपाईंका commits मा आधारित काम गरेको छ भने, तिनीहरूले तपाईंको commits लाई फेरि re-merge गर्नुपर्नेछ, जसले गर्दा अराजक र जटिल मर्जहरूको समस्या उत्पन्न हुन्छ।

जब तपाईं rebase गर्नुहुन्छ, तपाईं मूल commits लाई नयाँ commits ले प्रतिस्थापित गर्दै हुनुहुन्छ, जुन समान तर भिन्न हुन्छ। यदि तपाईंले commits लाई साझा रिपोजिटरीमा push गर्नुभयो र अरुले तिनीहरूको काम त्यसमा आधारित गरी अघि बढाएका छन् भने, ती commits लाई rebase गर्दा अरुलाई आफ्नो काम पुनः merge गर्नुपर्नेछ, र यसले गडबड र जटिल मर्ज समस्याहरू सिर्जना गर्न सक्छ।

मानौँ तपाईंले एक केन्द्रीय सर्भरबाट क्लोन गर्नुभयो र त्यसपछि त्यहाँबाट काम गर्न थाल्नुभयो। तपाईंको commit इतिहास यसरी देखिन सक्छ, तर यदि तपाईंले अरुले तपाईंका commits मा आधारित काम गरेपछि rebase गर्नुहुन्छ भने, यसले तपाईं र तपाईंका सहकार्यकर्ताहरूलाई समस्यामा पार्न सक्छ।

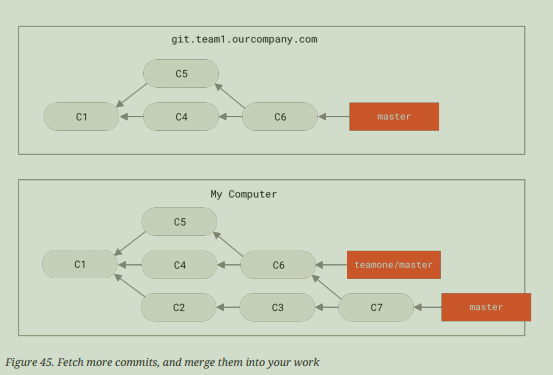
A screenshot of a computer

Description automatically generated

अब, कसैगरी अर्को व्यक्तिले थप काम गर्छ, जसमा merge पनि समावेश छ, र त्यसलाई केन्द्रीय सर्भरमा push गर्दछ। तपाईंले त्यो काम fetch गर्नुभयो र नयाँ remote branch लाई आफ्नो काममा merge गर्नुभयो। त्यसपछि तपाईंको commit history केही यसरी देखिन्छ:

तपाईंले fetch र merge गरेपछि commit historyमा नयाँ remote काम (merge सहित) सामेल हुन्छ।

यसले अझै प्रस्ट गर्दछ कि rebasing गर्दा shared repository मा आधारित commits लाई सावधानीपूर्वक व्यवस्थापन गर्नुपर्छ।

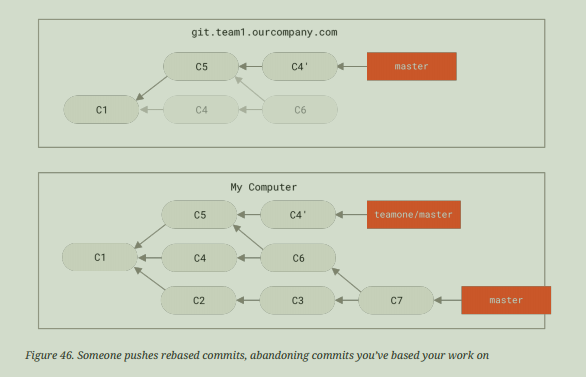


अहिले, जसले merged काम push गरेका थिए, तिनले आफ्नो कामलाई फेरी rebase गर्ने निर्णय गर्छन्। त्यसपछि उनीहरू

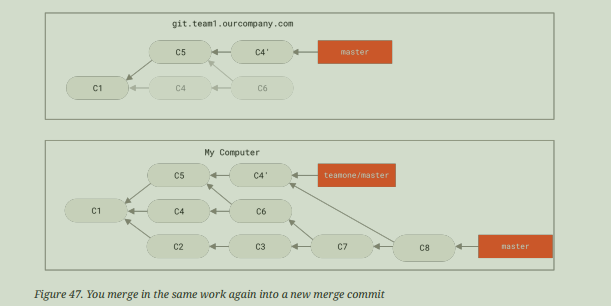
git push --force प्रयोग गरेर server मा भएको इतिहासलाई **overwrite** गर्छन्।

तपाईंले त्यस **server** बाट फेरि **fetch** गर्नुभयो, जसले गर्दा नयाँ **commits** तपाईंको लोकल रिपोजिटरीमा आउँछ।

यसले **shared repository** को इतिहास परिवर्तन गर्दा हुने जटिलता देखाउँछ। git push --force ले इतिहासलाई बदल्ने भएकाले, अरूलाई **conflict** समाधान गर्नुपर्छ र **commit** हिस्ट्री गडबड हुन सक्छ। यसकारण, सार्वजनिक रिपोजिटरीमा आधारित **commits** लाई **rebase** गर्दा विशेष सावधानी अपनाउनु आवश्यक हुन्छ।



अब तपाईंहरू दुबै समस्यामा पर्नुहुन्छ। यदि तपाईं git pull गर्नुहुन्छ भने, यो merge commit सिर्जना गर्छ, जसले दुबै हिस्ट्री लाइनहरू समावेश गर्दछ। त्यसपछि तपाईंको रिपोजिटरीको commit history केही यसरी देखिनेछ:



यो परिस्थितिमा, इतिहास गडबड हुने र conflict समाधान गर्नुपर्ने सम्भावना धेरै हुन्छ। यसले कामलाई अनावश्यक रूपमा जटिल बनाउँछ। यही कारणले, सार्वजनिक रिपोजिटरीमा आधारित commits लाई rebase नगर्न सुझाव दिइन्छ।

यदि तपाईंको commit history यस अवस्थामा छ भने, git log चलाउँदा तपाईंले एउटै author, date, र message भएका दुई commits देख्नुहुनेछ, जसले अन्योल उत्पन्न गर्न सक्छ।

त्यसबाहेक, यदि तपाईंले यो इतिहासलाई server मा फेरि push गर्नुभयो भने, rebased commits (जस्तै C4 र C6) पुन: central server मा देखिन्छन्। यसले थप गडबड उत्पन्न गर्न सक्छ।

यो स्पष्ट छ कि अर्को developer ले C4 र C6 लाई इतिहासमा नराख्न चाहन्थे, त्यसैले उनीहरूले rebase गरेका थिए। यस्तो स्थितिमा, rebasing गर्दा सावधानी अपनाउनु अत्यावश्यक हुन्छ, विशेष गरी shared repository को commits लाई।

Rebase गर्दा के गर्ने? - - -

यदि तपाईं यस्तो अवस्थामा पर्नुहुन्छ जहाँ तपाईंको टीमको कसैले force push गरेर commits लाई overwrite गरेको छ र तपाईंले त्यसमाथि आधारित भएर काम गर्नुभएको छ, Git मा केही सहायक सुविधाहरू छन् जसले तपाईंलाई मद्दत गर्न सक्छ। तपाईंको चुनौती भनेको आफ्नो काम र उनीहरूले फेरि लेखेको commits बीच छुट्याउनु हो।

Git मा commit को SHA-1 checksum सँगै, commit ले ल्याएको patch मा आधारित अर्को checksum पनि हुन्छ, जसलाई "patch-id" भनिन्छ। यो Git लाई uniquely तपाईंको काम पत्ता लगाउन र नयाँ शाखामा फेरि लागू गर्न मद्दत गर्छ।

**उदाहरणको समाधान:**

अघिल्लो परिदृश्यमा, तपाईंले merge नगरीकन git rebase teamone/master चलाउनु भयो भने, Git ले निम्न कार्यहरू गर्छ:

1. **हाम्रो शाखामा भएको अद्वितीय काम पत्ता लगाउनेछ**  
   (C2, C3, C4, C6, C7)
2. **Merge commits लाई छुट्याउनेछ**  
   (C2, C3, C4)
3. **Target branch मा पहिले नै समावेश नगरिएका commits पत्ता लगाउनेछ**
   * C4 पहिले नै C4' जस्तै छ, त्यसैले यो समावेश हुँदैन।
   * बाँकी C2 र C3 मात्र अद्वितीय commits हुन्।
4. **यी अद्वितीय commits लाई नयाँ शाखाको शीर्षमा लागू गर्नेछ।**

**परिणाम:**

merge commit को सट्टा, तपाईंको commits नयाँ शाखामा सही तरिकाले लागू हुन्छ, र इतिहास सफा र बुझ्न सजिलो देखिन्छ।  
यसले सुनिश्चित गर्छ कि तपाईंको अद्वितीय काम बचाइन्छ र कुनै अनावश्यक गडबडी हुँदैन।

A screenshot of a computer

Description automatically generated

**Rebase गर्दा सावधानी अपनाउनुपर्ने कुराहरू:**

* **C4 र C4' मिल्नुपर्छ:**  
  यो विधि काम गर्नका लागि तपाईंको पार्टनरले बनाएको C4' र तपाईंको C4 लगभग एउटै patch हुनुपर्छ। यदि दुई फरक छन् भने, Rebase ले यसलाई duplicate ठानेर अर्को C4-जस्तै patch थप्न सक्छ, जसले conflict गर्न सक्छ।
* **Rebase गर्दा git pull --rebase प्रयोग गर्नुहोस्:**  
  साधारण git pull को सट्टा git pull --rebase चलाउनुहोस्। यसले नयाँ commits लाई सही हिसाबले राख्छ।  
  वैकल्पिक रूपमा, तपाईंले पहिले git fetch र त्यसपछि git rebase teamone/master गर्न सक्नुहुन्छ।
* **Rebase default बनाउने:**  
  git pull लाई सँधै --rebase सहित चलाउन, तपाईं यो configuration सेट गर्न सक्नुहुन्छ:

$git config --global pull.rebase true

* **कुन अवस्थामा Rebase ठीक छ?:**
  + यदि commits तपाईंको कम्प्युटरमा मात्र छन् र कहिल्यै public push भएको छैन भने, Rebase सुरक्षित छ।
  + यदि commits push गरिएको छ तर कसैले पनि तिनलाई आधार मानेर काम गरेको छैन भने, Rebase गर्न सकिन्छ।
* **सावधानः Rebase पछि के हुन सक्छ?:**  
  यदि तपाईंले public commits लाई Rebase गर्नुभयो भने, र मानिसहरूले ती commits मा आधारित भएर काम गरेका छन् भने, यसले ठूलो समस्या र गडबडी ल्याउन सक्छ। यसले तपाईंका सहकर्मीहरूको निराशा र आलोचना निम्त्याउन सक्छ।
* **सहज बनाउन सबैलाई सूचित गर्नुहोस्:**  
  यदि तपाईं वा तपाईंको पार्टनरले Rebase गरिसकेपछि commits push गर्नुपर्छ भने, सबैलाई git pull --rebase चलाउन सूचित गर्नुहोस्। यसले गडबडी कम गर्न मद्दत गर्छ।

## Rebase र Merge को तुलना

**Rebase** र **Merge** बीच छनोट गर्दा तपाईंको प्रोजेक्टको इतिहासलाई कस्तो प्रकारले हेर्नुहुन्छ भन्ने कुरामा भर पर्छ।

1. **Commit History लाई घटनाको रेकर्डको रूपमा हेर्ने दृष्टिकोण**
   * **Merge**: यस दृष्टिकोणमा, तपाईंको commit इतिहास भनेको **के भएको थियो भन्ने ऐतिहासिक रेकर्ड** हो। Merge commits ले प्रत्येक घटना र सन्दर्भलाई दर्शाउँछन्, जसमा सबै अव्यवस्थित डिटेल र शाखाहरू समावेश छन्। इतिहासलाई फेरि लेख्न (जस्तै rebasing गर्दा) यो साँचो घटित भएको कुरा लुकाउनु जस्तो लाग्न सक्छ। यदि merge commits अलि अव्यवस्थित छन् भने पनि, ती नै सही तरिकाले देखाउँछन् कि घटना कस्तो क्रममा भयो।
2. **Commit History लाई परियोजना निर्माणको एक सुसंगत कथाको रूपमा हेर्ने दृष्टिकोण**
   * **Rebase**: अर्को पक्षमा commit इतिहासलाई **परियोजना कसरी बन्यो भन्ने कथा** को रूपमा हेरिन्छ। यदि तपाईं पुस्तकको पहिलो ड्राफ्ट देखाउँदैन भने, प्रोजेक्टको इतिहासको सबै गल्ती र dead-end हरू पनि देखाउनु भनेको के हो र? Rebase ले इतिहासलाई सफा पार्न मद्दत गर्दछ, यसले थप merge commits हटाएर अधिक रेखीय र अनुसरण गर्न सजिलो बनाउँछ। यसरी तपाईं प्रोजेक्टको विकासको स्वच्छ र स्पष्ट कथा प्रस्तुत गर्न सक्नुहुन्छ।

**कुन उत्तम हो?**

* **यसमा सरल उत्तर छैन**। दुबै विधिहरूका आफ्ना फाइदाहरू र कमजोरीहरू छन्, र कुन विधि उपयुक्त छ भन्ने कुरा सन्दर्भमा भर पर्छ। Git एक शक्तिशाली उपकरण हो, जसले तपाईंलाई धेरै कुरा गर्न र गर्न दिन्छ, र यसमा तपाईं र तपाईंको टिमलाई कुन तरिका उत्तम लाग्छ भन्ने कुरा चयन गर्नु पर्छ।

**दुवै संसारको सर्वोत्तम:**

* तपाईंले **स्थानीय परिवर्तनहरूलाई rebase गर्न सक्नुहुन्छ** र त्यसपछि ती परिवर्तनहरूलाई सफा र सुसंगत बनाउन सक्नुहुन्छ।
* **पहिले नै push गरेको commit हरूलाई कहिल्यै rebase नगर्नुहोस्**, कारण यसले अरु सहकार्यकर्ताहरूलाई समस्या दिन सक्छ जसले ती commits मा आधारित काम गरेका छन्।

------------- END ------------------