

मूलभूत Branching र Merging

Git मा Branching र Merging महत्वपूर्ण अवधारणाहरू हुन्। यिनीहरूको सहायताले तपाईं आफ्नो विकास प्रक्रिया कसरी सहज बनाउन सक्नुहुन्छ भनेर एक वास्तविक उदाहरण मार्फत सिक्नें।

परिदृश्य: वेबसाइट विकास गर्नुहोस्

तपाईं एउटा वेबसाइटमा काम गर्दै हुनुहुन्छ, र यी कामहरू गर्नुपर्नेछ:

चरणहरू:

1. **वेबसाइटमा काम सुरु गर्नुहोस्:**
आफ्नो प्रोजेक्टको main शाखामा अपडेट र परिवर्तनहरू गर्नुहोस्।
2. **नयाँ Feature को लागि Branch बनाउनुहोस्:**
तपाईंले काम गर्दै गरेको नयाँ User Story को लागि एउटा नयाँ Branch बनाउनुहोस्।
3. **Branch मा काम गर्नुहोस्:**
नयाँ Feature को लागि परिवर्तनहरू गर्नुहोस् र Commit गर्नुहोस्।

हटफिक्सको आवश्यकता:

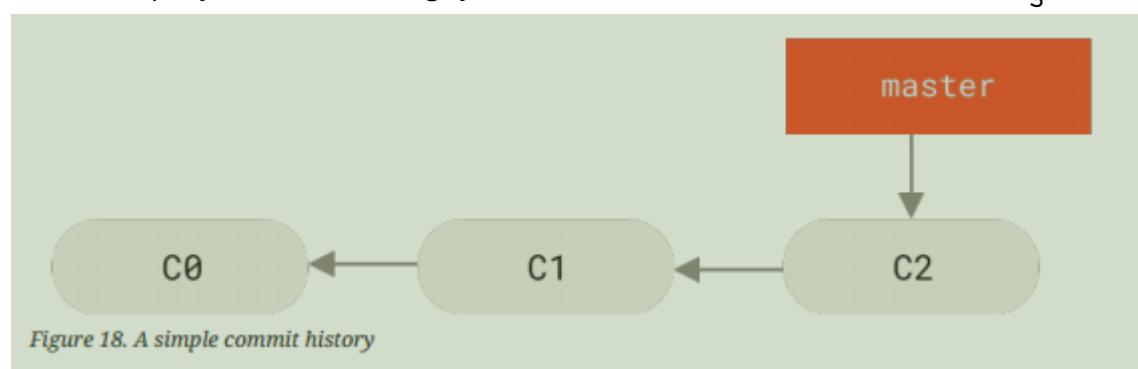
यो अवस्थामा, तपाईंलाई अचानक एउटा Call आउँछ कि अर्को Issue अत्यन्तै जरुरी छ र तपाईंलाई Hotfix तयार गर्नुपर्नेछ।

चरणहरू:

1. **Production Branch मा Switch गर्नुहोस्:**
मुख्य Production Branch मा जानुहोस्।
2. **Hotfix को लागि नयाँ Branch बनाउनुहोस्:**
Hotfix को लागि अलग Branch बनाउनुहोस्।
3. **Testing पछि Merge गर्नुहोस्:**
Hotfix Branch लाई परीक्षण गरेपछि Production मा Merge गर्नुहोस् र Push गर्नुहोस्।
4. **फेरी आफ्नै काममा फर्किनुहोस्:**
Hotfix सम्पन्न भएपछि, आफ्नो User Story को Branch मा फर्किनुहोस् र काम जारी राख्नुहोस्।

मूलभूत Branching - - -

सबैभन्दा पहिले, मानौं तपाईं आफ्नो प्रोजेक्टमा काम गर्दै हुनुहुन्छ र master branch मा केही commits गरिसक्नुभएको छ। अब तपाईंले आफ्नो company को issue-tracking system मा रहेको issue #53 मा काम गर्ने निर्णय गर्नुभएको छ।



नयाँ Branch सिर्जना र Switch गर्ने

नयाँ Branch सिर्जना गरेर त्यसमा एकेचोटि Switch गर्नका लागि, तपाईंले git checkout command लाई -b switch को साथ प्रयोग गर्न सक्नुहुन्छ।

उदाहरण:

```
$ git checkout -b issue-53
```

यो command ले के गर्छे?

1. नयाँ Branch issue-53 सिर्जना गर्छ।
2. HEAD pointer लाई नयाँ Branch मा Switch गरिदिन्छ।

अब, तपाईं नयाँ Branch issue-53 मा हुनुहुन्छ, जहाँ तपाईंले आफ्नो कामलाई अलग रूपमा जारी राख्न सक्नुहुन्छ।

नयाँ Branch सिर्जना र Switch गर्ने (Short Command) - - -

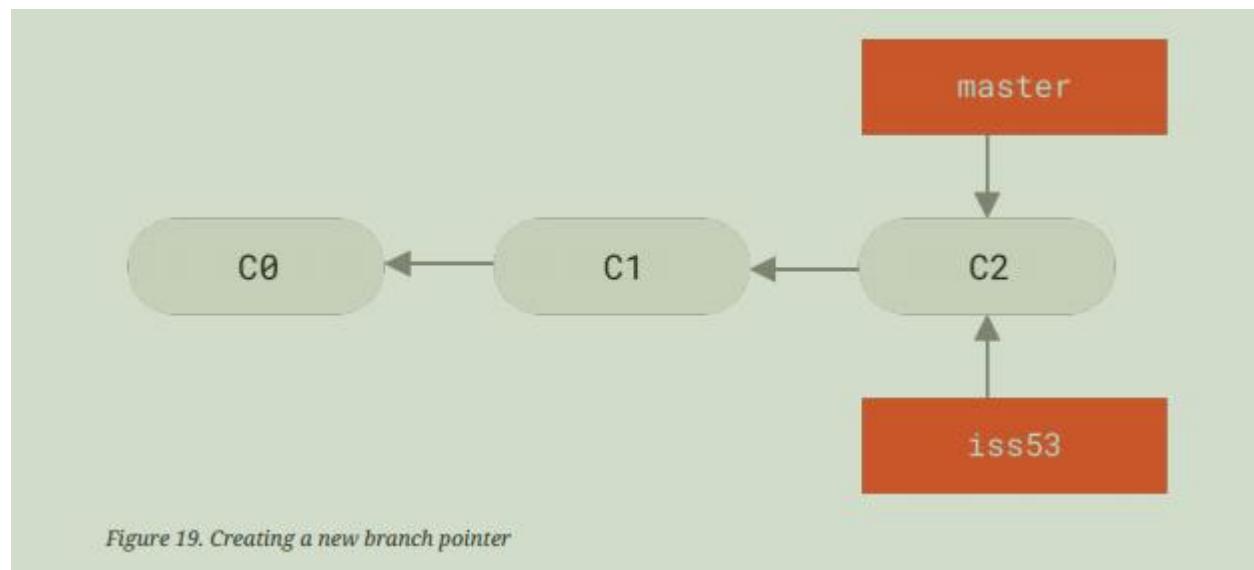
तपाईंले नयाँ Branch सिर्जना गरेर त्यसमा Switch गर्ने निम्न command प्रयोग गर्न सक्नुहुन्छ:

```
$ git checkout -b iss53
```

```
Switched to a new branch 'iss53'
```

यो command ले दुईवटा काम एकेचोटि गर्छे:

1. iss53 नामको नयाँ Branch सिर्जना गर्छ।
2. HEAD pointer लाई iss53 Branch मा Switch गर्छ।



यो commandको लामो संस्करण:

```
$ git branch iss53      # नयाँ Branch सिर्जना गर्छ।
```

```
$ git checkout iss53    # त्यस Branch मा Switch गर्छ।
```

किन प्रयोग गर्ने?

git checkout -b छोटकरीमा Branch सिर्जना गर्ने र Switch गर्नका लागि प्रयोग हुन्छ। यसले समय बचत गर्छ र प्रक्रिया सरल बनाउँछ।

अब तपाईं iss53 Branch मा हुनुहुन्छ र यसमा अलग-अलग feature वा fixes मा काम गर्न सक्नुहुन्छ।

अब तपाईं आफ्नो वेबसाइटमा काम गर्न सक्नुहुन्छ र commits थप्न सक्नुहुन्छ। जब तपाईं commits गर्नुहुन्छ, iss53 Branch अगाडि बढ्छ किनकि तपाईंले यसलाई checkout गरिसक्नु भएको छ (HEAD pointer अहिले यसै Branch मा छ)।

Commands:

```
$ vim index.html  
$ git commit -a -m 'Create new footer [issue 53]'
```

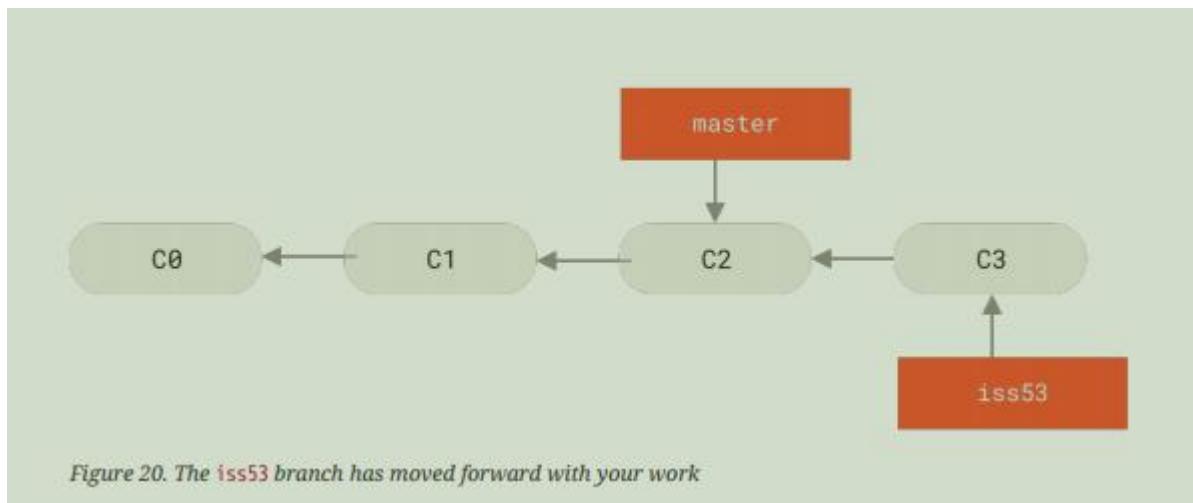


Figure 20. The iss53 branch has moved forward with your work

tt

यसले के गर्छ?

1. **vim index.html:** यो command प्रयोग गरेर तपाईंले आफ्नो वेबसाइटको index.html फाइलमा आवश्यक परिवर्तन गर्न सक्नुहुन्छ।
2. **git commit -a -m 'Create new footer [issue 53]:'**
 - -a: यो flag ले सबै modified files (tracked files मात्र) commit मा समावेश गर्न मद्दत गर्छ।
 - -m: commit message दिन प्रयोग हुन्छ।
 - 'Create new footer [issue 53]': यो commit message हो जसले तपाईंको परिवर्तनको विवरण दिन्छ।

परिणाम:

यो commit गर्दा iss53 Branch अगाडि बढ्छ, किनकि HEAD pointer अहिले iss53 मा छ। यो branch मा तपाईंले समस्या #53 (issue 53) लाई सम्बोधन गर्दै नयाँ footer सिर्जना गर्नुभएको छ। अब तपाईं थप commits गरेर आफ्नो काम पूरा गर्न सक्नुहुन्छ!

अब तपाईंलाई website मा समस्या आएको call प्राप्त हुन्छ, र यो समस्या तुरुन्तै समाधान गर्नुपर्ने हुन्छ। Git को मद्दतले, तपाईंले आफ्नो iss53 मा गरिएका परिवर्तनहरू बिना नै fix deploy गर्न सक्नुहुन्छ। त्यसका लागि तपाईंले आफ्नो master branch मा फर्किनु पर्छ।

तर, ध्यान दिनुहोस्:

यदि तपाईंको **working directory** वा **staging area** मा uncommitted changes छन् र ती changes checkout गर्न लागेको branch conflict गर्छन् भने, Git ले तपाईंलाई branch switch गर्न दिँदैन। त्यसैले, branch switch गर्नु अघि तपाईंको working state **clean** हुनु आवश्यक छ।

Clean Working State प्राप्त गर्न सुझाव:

1. **Commit** गर्नुहोस्: सबै परिवर्तनलाई commit गर्नुहोस्।
2. **Stash** गर्नुहोस्: अस्थायी रूपमा परिवर्तनलाई सुरक्षित राख्न।
3. **Amend Commit** गर्नुहोस्: यदि पहिले commit गर्ने छुटेका फाइलहरू छन् भने commit संशोधन गर्नुहोस्।

Command to Switch Back to Master Branch:

```
$ git checkout master
```

के हुन्छ?

- **git checkout master** चलाउँदा:
 - तपाईंको HEAD pointer master branch मा सर्छ।
 - Working directory पनि master को snapshot अनुसार revert हुन्छ।

अब तपाईंले production मा भएको issue समाधान गर्न आफ्नो master branch बाट काम सुरु गर्न सक्नुहुन्छ।

Hotfix Branch बनाएर समस्या समाधान गर्नुहोस्----

अब तपाईं आफ्नो project मा hotfix गर्नका लागि branch बनाउनु पर्छ। Hotfix branch मा काम गर्दा, Git तपाईंको working directory लाई अटोमेटिक रूपले परिवर्तन गर्दछ ताकि branch को अन्तिम commit जस्तै देखियोस्।

Command to Switch to Master Branch:

```
$ git checkout master  
Switched to branch 'master'
```

के हुन्छ?

- Project को working directory master branch को अन्तिम commit को अवस्थामा revert हुन्छ।
- तपाईंले issue #53 मा काम सुरु गर्नु अघि project जस्तो थियो, उस्तै देखिन्छ।

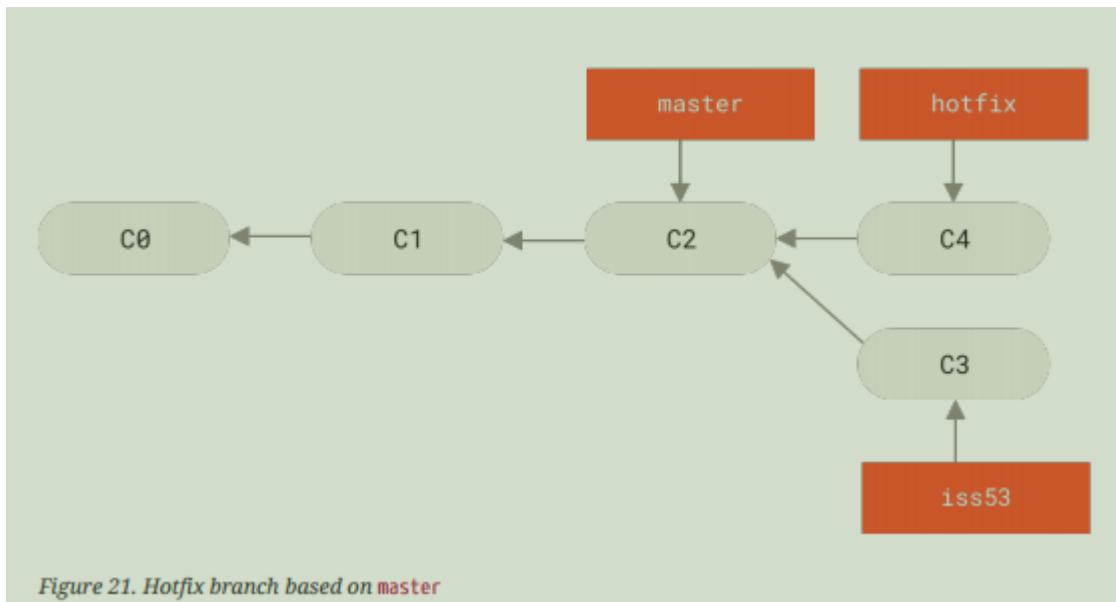


Figure 21. Hotfix branch based on master

Hotfix Branch बनाउने र Commit गर्ने:

अब hotfix गर्नको लागि नयाँ branch बनाउनुहोस्:

```
$ git checkout -b hotfix  
Switched to a new branch 'hotfix'
```

Hotfix Commit गर्ने:

1. File मा समस्या समाधान गर्न code परिवर्तन गर्नुहोस्:

```
$ vim index.html
```

2. Commit गर्ने:

```
$ git commit -a -m 'Fix broken email address'  
[hotfix 1fb7853] Fix broken email address  
1 file changed, 2 insertions(+)
```

के भयो?

- **Hotfix Branch:** तपाईंको नयाँ branch hotfix मा commit भयो।
- Commit Message: 'Fix broken email address' ले तपाईंको परिवर्तनको विवरण दिन्छ।
- **File Changes:** 1 फाइलमा 2 नयाँ लाइन थपिए।

अब तपाईंले यो hotfix branch मा काम गरिसक्नुभएको छ, र यो branch पछि production मा merge गर्न सकिन्छ।

अब तपाईंले आफ्नो hotfix को testing गर्न सक्नुहुन्छ, सुनिश्चित गर्नुहोस् कि hotfix सही छ, र अन्ततः hotfix branch लाई master branch मा merge गरेर production मा deploy गर्न सक्नुहुन्छ।

Hotfix Merge गर्ने चरणहरू:

1. Master Branch मा Switch गर्नुहोस्:

```
$ git checkout master  
Switched to branch 'master'
```

2. Hotfix Merge गर्नुहोस्:

```
$ git merge hotfix  
Updating f42c576..3a0874c  
Fast-forward  
 index.html | 2 ++  
 1 file changed, 2 insertions(+)
```

के भयो?

- **Fast-forward Merge:**

- hotfix branch को commit (C4) सिधै master branch को commit (C2) भन्दा अगाडि थियो।
- यस्तो अवस्थामा, Git ले कुनै conflict resolve गर्न नपर्ने भएकाले pointer सिधै अगाडि बढाउँछ। यसलाई "**fast-forward**" merge भनिन्छ।

- **File Changes:**

- Merge पछि, 1 फाइल (index.html) मा 2 नयाँ लाइन थपिएका छन्।

Result:

- Hotfix को परिवर्तन अब master branch मा commit भएको छ।
- Master branch को snapshot (C4) मा hotfix समावेश छ।
- तपाईंको fix production मा deploy गर्न तयार छ। 🎉

Note:

Fast-forward merge केवल तब हुन्छ जब merging branches बीच कुनै divergent history हुँदैन।

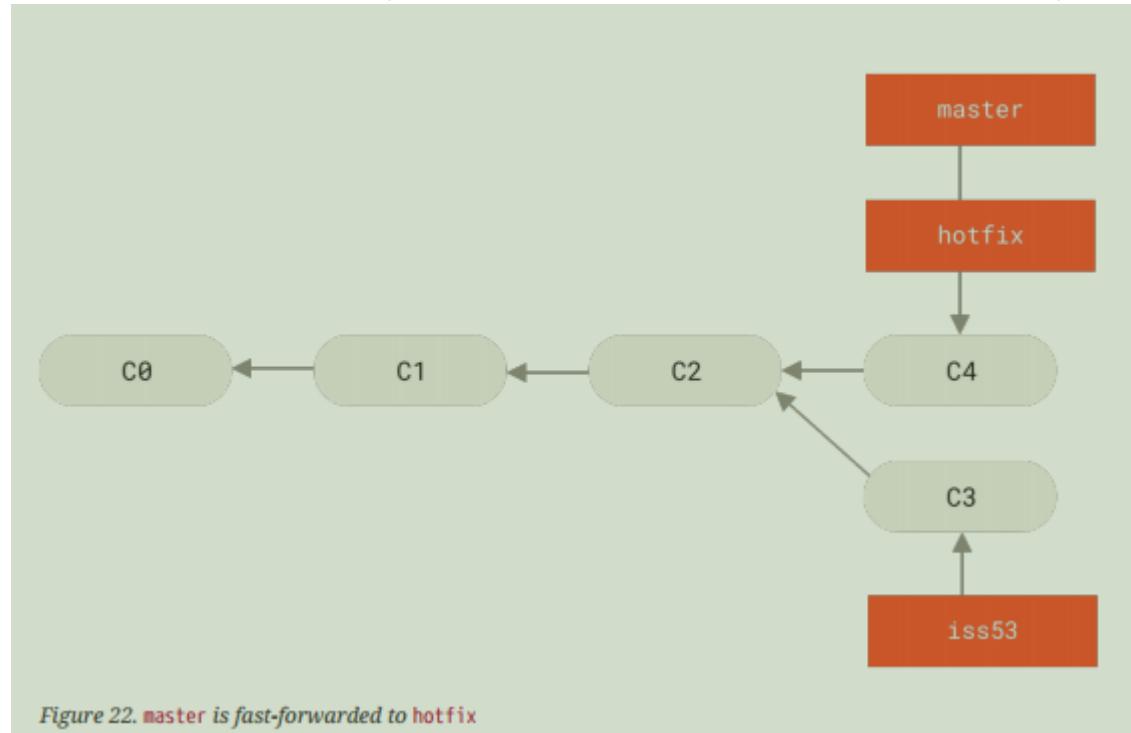


Figure 22. master is fast-forwarded to hotfix

तपाईंको super-important fix deploy भइसकेपछि, अब तपाईं आफ्नो पहिलाको काममा फर्किन सक्नुहुन्छ। तर, त्योभन्दा पहिले, hotfix branch लाई delete गर्नु आवश्यक छ किनकि master branch पहिले मै hotfix commit (3a0874c) मा point गर्दैछ।

Hotfix Branch Delete गर्ने तरिका:

1. Hotfix Branch Delete गर्नुहोस्:

```
$ git branch -d hotfix
Deleted branch hotfix (3a0874c).
```

- के भयो?

- hotfix branch अब delete भएको छ।
- Master branch ले hotfix commit (3a0874c) समावेश गरिसकेकोले, यो branch अब आवश्यक छैन।

Work-in-Progress Branch मा फर्कनुहोस्:

2. Issue #53 को Branch मा Switch गर्नुहोस्:

```
$ git checkout iss53
Switched to branch "iss53"
```

3. काम जारी राख्नुहोस्:

- Code edit गर्नुहोस् र commit गर्नुहोस्:

```
$ vim index.html
$ git commit -a -m 'Finish the new footer [issue 53]'
[iss53 ad82d7a] Finish the new footer [issue 53]
1 file changed, 1 insertion(+)
```

के भयो?

- Hotfix Branch हटाइयो:
 - अब unnecessary branch हटाएर repository सफा गरिएको छ।
- काम जारी राखियो:
 - iss53 branch मा switch गरेर footer design को काम पूरा गरियो।

Final State:

- Master branch मा hotfix merge भइसकेको छ।
- iss53 branch मा issue #53 को काम पूरा हुँदैछ।
- Repository organized र काम streamlined छ।

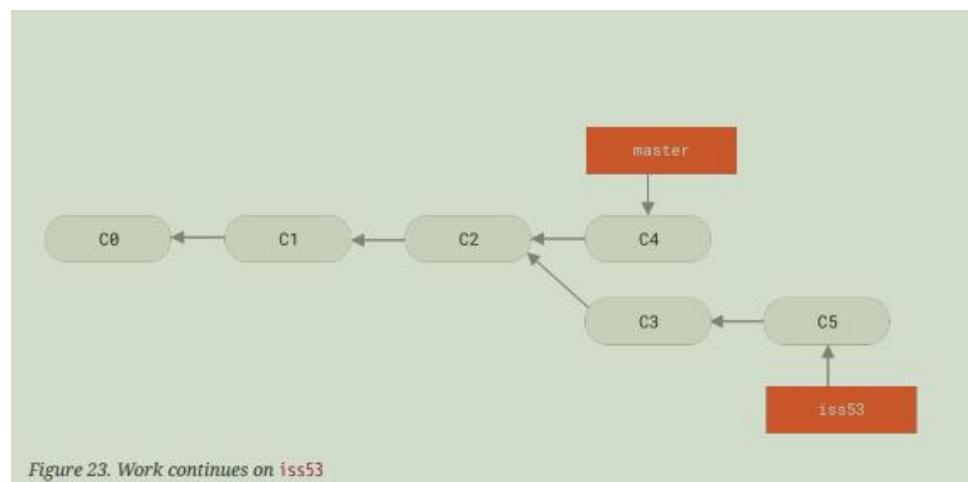


Figure 23. Work continues on iss53

यहाँ यो कुरा ध्यान दिन योग्य छ कि तपाईंले hotfix branch मा गरेको काम iss53 branch मा समावेश हैन। यदि तपाईंलाई hotfix changes लाई iss53 branch मा ल्याउन आवश्यक छ भने, तपाईंले **master** branch लाई **iss53** branch मा merge गर्न सक्छौं। यसले hotfix changes लाई iss53 branch मा समावेश गर्दछ।

Hotfix Changes लाई iss53 मा Merge गर्ने तरिका:

1. Iss53 Branch मा Switch गर्नुहोस्:

```
$ git checkout iss53
```

2. Master Branch लाई Iss53 मा Merge गर्नुहोस्:

```
$ git merge master
```

के हुनेछ?

- Merge Process:

- Git ले master branch मा भएका सबै परिवर्तनहरूलाई iss53 branch मा merge गर्नेछ।
- यदि कुनै conflicts भए भने, Git ले तपाईंलाई त्यसको समाधान गर्न भनेको छ।

Alternative:

- यदि तपाईं पछि iss53 branch लाई **master** मा merge गर्न चाहनुहुन्छ भने, तपाईं त्यसको लागि पर्खन सक्नुहुन्छ। त्यसपछि सबै कामहरू एकसाथ master branch मा ल्याउन सक्नुहुन्छ।

तपाईंको Workflow:

- अहिले तपाईं hotfix changes लाई iss53 मा merge नगरी iss53 को काम जारी राख्न सक्नुहुन्छ र पछि जब तपाईं iss53 लाई **master** मा merge गर्नुहुन्छ, तब यो सब एकसाथ समावेश हुनेछ।
- यसले तपाईंलाई task हरु राम्रोसँग manage गर्न मद्दत पुर्योउँछ।

Break -1 -----

Basic Merging

अब तपाईंले issue #53 को काम पूरा गर्नुभएको छ, अब iss53 ब्रान्चलाई master ब्रान्चमा मर्ज गर्ने समय आएको छ। मर्ज गर्दा के हुन्छ भन्ने बारेमा विस्तृत व्याख्या यहाँ छ।

iss53 लाई master मा मर्ज गर्ने कदमहरू:

1. **master** ब्रान्चमा स्विच गर्नुहोस्: पहिले, तपाईंको master ब्रान्चमा जानुहोस्, किनकि तपाईं iss53 को परिवर्तनहरू त्यहीमा मर्ज गर्न चाहनुहुन्छ।

```
$ git checkout master
```

```
Switched to branch 'master'
```

2. **iss53** लाई **master** मा मर्ज गर्नुहोस्: git merge कमाण्ड चलाउनुहोस्, जसले iss53 का परिवर्तनहरू master मा मर्ज गर्छ:

```
$ git merge iss53
```

```
Merge made by the 'recursive' strategy.
```

```
index.html | 1 +
```

```
1 file changed, 1 insertion(+)
```

मर्ज गर्दा के भयो?

- **मर्ज स्ट्राटेजी:**

- "Merge made by the 'recursive' strategy" भन्ने सन्देशले देखाउँछ कि Git ले डिफल्ट recursive merge strategy प्रयोग गर्यो। यो मर्ज गर्ने सामान्य तरिका हो जब दुई ब्रान्च बीचमा धेरै परिवर्तनहरू हुन्छन् र ती ब्रान्चहरू बीचको इतिहास फरक हुन्छ।

- **थ्री-वे मर्ज:**

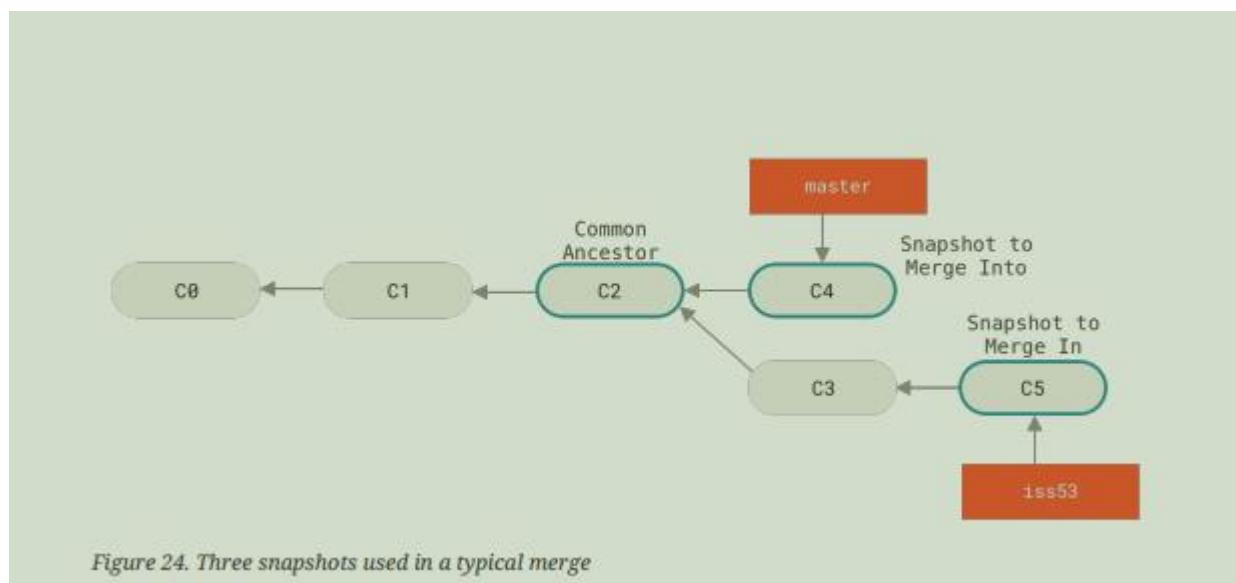
- master र iss53 बीचमा सिधा सम्बन्ध नभएको कारण, Git ले थ्री-वे मर्ज गर्छ।
 - **Snapshot 1:** तपाईंको master ब्रान्चको पछिलो कमिट।
 - **Snapshot 2:** iss53 ब्रान्चको पछिलो कमिट।
 - **Snapshot 3:** ती दुई ब्रान्चहरूको साझा पूर्वज कमिट, जहाँ तिनीहरूले फरक हुँदै जान थाले।
- Git यी तीनवटै snapshots लाई तुलना गरेर परिवर्तनहरू मर्ज गर्दछ।

- **परिणाम:**

- Git ले स्वचालित रूपमा परिवर्तनहरू मर्ज गरेर master ब्रान्चमा लागू गर्दछ।
- index.html फाइलमा परिवर्तन गरिएको छ, जसले सन्देश "index.html | 1 +" देखाउँछ (यो 1 नयाँ लाइन थपिएको हो भन्ने जनाउँछ)।

Hotfix मर्जसँग तुलना गर्दा भिन्नता?

- **Hotfix मर्ज** मा, तपाईंले fast-forward merge गर्नुभएको थियो, जहाँ परिवर्तनहरू सिधा अगाडि थिए र कुनै पनि भिन्नता थिएन।
- यस अवस्थामा, ब्रान्चहरू बीचमा भिन्नता रहेकोले Git ले तीनवटै snapshots बीचको भिन्नता मिलाएर मर्ज गर्नको लागि काम गरेको हो।



कार्यप्रवाहको सारांश:

- तपाईंले सफलतापूर्वक iss53 ब्रान्चका परिवर्तनहरू master ब्रान्चमा मर्ज गर्नुभयो र Git ले तीन-तरिकामा मर्ज गर्यो।
- अब तपाईं master ब्रान्चको परीक्षण र डिप्लोइ गर्न तयार हुनुहुन्छ यदि सबै कुरा ठीक छ भने।

यस प्रक्रिया मार्फत तपाईंका विभिन्न ब्रान्चहरू (जस्तै iss53 र hotfix) को परिवर्तनहरूलाई Git ले सही तरिकामा मर्ज गरेर मिलाउँछ।

जब तपाईं दुईवटा शाखा (branches) बीच मर्ज गर्नुहुन्छ, Git केवल शाखाको पोइन्टरलाई अगाडि बढाउँदैन, जस्तो कि फास्ट-फरवर्ड मर्जमा गरिन्छ। बरु, Git एक नयाँ स्न्यापशट सिर्जना गर्छ जुन दुबै शाखाबाट आएको परिवर्तनलाई मिलाएर तयार गरिएको हुन्छ। यस नयाँ स्न्यापशटलाई मर्ज कमिट भनिन्छ, जुन विशेष हुन्छ किनकि यसका दुईवटा पेरेंट्स (parents) हुन्छन्।

यसको महत्त्व के हो?

- मर्ज कमिट:** मर्ज कमिट भनेको एक यस्तो कमिट हो जसले दुईवटा वा सोभन्दा बढी शाखाबाट आएको परिवर्तनलाई मिलाएर तयार गरिन्छ। यसले मर्ज गरिएका शाखाहरूको इतिहासलाई लिंक गर्छ र "ज्वाइन प्वाइन्ट" को रूपमा काम गर्छ। यो कमिटसँग दुई पेरेंट्स हुन्छन् – एउटा तपाईंको मर्ज गरिरहेको शाखाको र अर्को मर्ज गरिएका शाखाको।

तिन-तर्फी मर्ज के हो?

तिन-तर्फी मर्जमा Git तीनवटा फरक बिन्दुलाई तुलना गर्छ:

- दुई शाखा विभाजन भएका सामान्य पूर्वज (common ancestor) को कमिट।
- तपाईंको वर्तमान शाखाको कमिट (जस्तै master शाखा)।
- तपाईं मर्ज गर्न लागेको शाखाको कमिट (जस्तै iss53 शाखा)।

Git यी तीनवटा बिन्दुलाई प्रयोग गरेर एक नयाँ स्न्यापशट सिर्जना गर्छ जसमा दुबै शाखाबाट आएका परिवर्तनहरू समावेश हुन्छ। यो नयाँ कमिटलाई मर्ज कमिट भनिन्छ, जसको दुई पेरेंट कमिटहरू हुन्छन्:

- पेरेंट १: तपाईंको वर्तमान शाखाको कमिट (जस्तै master)।
- पेरेंट २: मर्ज गरिएको शाखाको कमिट (जस्तै iss53)।

उदाहरण:

यहाँ एक साधारण उदाहरण छ जुन के हुन्छ भन्ने कुरा स्पष्ट गर्न:

- तपाईं master शाखामा हुनुहुन्छ, र तपाईं iss53 शाखासँग मर्ज गर्नुहुन्छ।
- Git सामान्य पूर्वजलाई तुलना गर्छ (जहाँ master र iss53 शाखाहरू अलग भएको थिए) र दुबै शाखामा भएका अन्तिम कमिटहरूको तुलना गर्छ।
- Git फरकहरू समाधान गर्छ र एउटा नयाँ स्न्यापशट सिर्जना गर्छ जुन दुबै शाखाबाट आएका परिवर्तनलाई समावेश गर्छ।
- Git त्यसपछि मर्ज कमिट सिर्जना गर्छ। यो मर्ज कमिट:
 - दुई पेरेंट्स हुन्छन् (एक master शाखाको अन्तिम कमिट र अर्को iss53 शाखाको अन्तिम कमिट)।
 - नयाँ स्न्यापशटलाई संकेत गर्छ जुन दुबै शाखाबाट आएका परिवर्तनहरू समावेश गर्दछ।
 - Git इतिहासमा यसरी देखिन्छ:

* Merge commit (दुई पेरेंट्स)

| \

| * * iss53 शाखाको कमिट

* | master शाखाको कमिट

| /

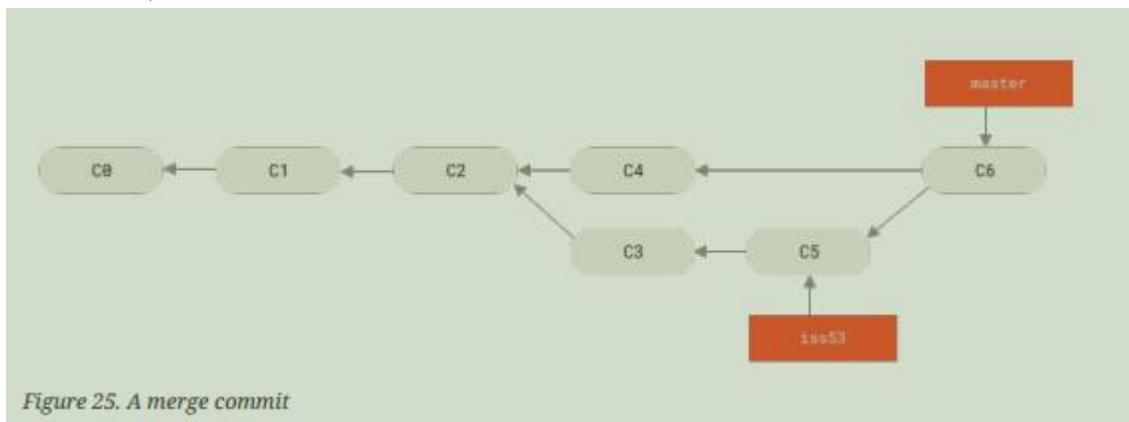


Figure 25. A merge commit

मर्ज कमिट विशेष के छ?

- धैरै पेरेंट्स: साधारण कमिटहरूमा एक मात्र पेरेंट (अर्थात् अधिल्लो कमिट) हुन्छ, जबकि मर्ज कमिटमा दुई पेरेंट्स हुन्छन्: एक तपाईंको शाखाबाट र अर्को मर्ज गरिएको शाखाबाट।
- इतिहासको प्रतिनिधित्व: मर्ज कमिटले दुई शाखाहरूलाई एकल विकास रेखामा एकीकृत गर्ने बिन्दुको रूपमा काम गर्छ। यसले मर्ज गरिएको शाखाहरूको इतिहासलाई संरक्षण गर्छ।

मर्ज कमिट पछि के हुन्छ?

मर्ज कमिट सिर्जना गरेपछि, Git master शाखालाई यो नयाँ कमिटमा अद्यावधिक गर्छ। अब दुबै शाखाको इतिहास जस्तै छ, र मर्ज कमिटले स्पष्ट रूपमा देखाउँछ कि कहाँ शाखाहरू मर्ज गरिएको थियो।

- मर्ज कमिटको उदाहरण:

```
$ git checkout master
$ git merge iss53
Merge made by the 'recursive' strategy.
index.html | 1 +
1 file changed, 1 insertion(+)
```

यस प्रक्रियाले Git लाई परियोजनाको इतिहासलाई संरचित तरिकाले दृश्यक गर्न मद्दत गर्छ, यहाँसम्म कि शाखाहरू विभाजन भइ पछि मर्ज गरेर एकजुट गरिएको छ।

अब तपाईंको काम मर्ज गरिएको छ र तपाईंलाई iss53 शाखाको अरु कुनै आवश्यक छैन। तपाईं आफ्नो इश्यू दृश्याकिंग प्रणालीमा इश्यू बन्द गर्न सक्नुहुन्छ र शाखा मेट्न सक्नुहुन्छ:

```
$ git branch -d iss53
Deleted branch iss53 (abc1234).
```

यस कमाण्डले iss53 शाखालाई मेट्नेछ। यदि तपाईंको शाखा मर्ज भएको छैन भने, Git तपाईंलाई मर्ज नगरेको शाखा मेट्न रोक्नेछ र यो चेतावनी दिनेछ:

```
$ git branch -d iss53
error: The branch 'iss53' is not fully merged.
If you are sure you want to delete it, run 'git branch -D iss53'.
```

त्यस अवस्थामा, तपाईं -D विकल्प प्रयोग गरेर शाखालाई मेट्न सक्नुहुन्छ:

```
$ git branch -D iss53
```

Deleted branch iss53 (abc1234).

यो प्रक्रिया पछि, तपाईंले **iss53** शाखा मेटिसकेको हुनेछ र तपाईं अब फोकस गर्न सक्नुहुन्छ आगामी काममा।

Basic Merge Conflicts

कधीकधी, यो प्रक्रिया सहजसँग अगाडि बढ्दैन। यदि तपाईंले मर्ज गरिरहेका दुई शाखामा एउटै फाइलको एउटै भागमा फरक फरक परिवर्तनहरू गर्नुभयो भने, Git ती परिवर्तनहरूलाई सफा रूपमा मर्ज गर्न सक्दैन। यदि तपाईंको **iss53** इश्यूको समाधानले **hotfix** शाखाको समान फाइलको एउटै भागलाई परिवर्तन गरेको छ भने, तपाईंलाई मर्ज कन्फिलक्ट देखिनेछ, जसको उदाहरण यसरी देखिन्छ:

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

यस अवस्थामा, Git स्वचालित रूपमा मर्ज गर्न सक्दैन, र तपाईंलाई कन्फिलक्ट भएको फाइल (यहाँ **index.html**)लाई म्यानुअली हल गर्ने आवश्यक हुनेछ। कन्फिलक्ट हल गर्नका लागि तपाईंले कन्फिलक्ट भएका फाइललाई खोल्नुहोस् र Git द्वारा चिन्ह लगाइएको क्षेत्रहरू (निस्सन्देह <<<<<, =====, र >>>>> का साथ) समेट्नुहोस्, र त्यसपछि कन्फिलक्टलाई सही गरेर फाइललाई अपडेट गर्नुहोस्।

उदाहरण:

```
<<<<< HEAD
This is the content from the current branch (master).
=====
This is the content from the other branch (iss53).
>>>>> iss53
```

तपाईंले यी क्षेत्रहरूलाई मर्ज गरी आफ्नो इच्छाएको समाधान लेरखुपर्छ, र पछि फाइललाई नयाँ कमिटमा सेभ गर्न सक्नुहुन्छ:

```
$ git add index.html
$ git commit -m "Resolved merge conflict in index.html"
```

यसपछि मर्ज कन्फिलक्ट हल गरिएको हुनेछ र तपाईंको Git इतिहास अपडेट हुनेछ।

Gitले स्वचालित रूपमा नयाँ मर्ज कमिट सिर्जना गरेको छैन। यसले प्रक्रिया रोकिएको छ र तपाईंलाई कन्फिलक्ट समाधान गर्नका लागि रोकिएको छ। यदि तपाईं मर्ज कन्फिलक्टपछि कुन फाइलहरू मर्ज हुन बाँकी छन् भनेर हेर्न चाहनुहुन्छ भने, तपाईं git status चलाउन सक्नुहुन्छ:

```
$ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")
Unmerged paths:
  (use "git add <file>..." to mark resolution)
    both modified: index.html
no changes added to commit (use "git add" and/or "git commit -a")
```

कुनै पनि फाइल जसमा मर्ज कन्फिलक्ट भएको छ र त्यो समाधान गरिएको छैन, तिनीहरूलाई unmerged भनिन्छ। Git कन्फिलक्ट भएको फाइलहरूमा स्ट्रिक्ट कन्फिलक्ट-रिजोल्युसन मार्करहरू थप्दछ, जसको मद्दतले तपाईंले ती फाइलहरू म्यानुअली खोल्न सक्नुहुन्छ र कन्फिलक्ट समाधान गर्न सक्नुहुन्छ। तपाईंको फाइलमा यस्तो खालको खण्ड हुनेछ:

```
<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
please contact us at support@github.com
</div>
>>>>> iss53:index.html
```

यसको अर्थ हो कि **HEAD** (तपाईंको **master** शाखा, किनकि तपाईंले मर्ज कमाण्ड चलाउँदा यो शाखा चेकआउट गरिएको थियो) को संस्करण त्यो ब्लकको माथिल्लो भाग हो (सारे सबै कुरा ===== भन्दा माथि), जबकि तपाईंको **iss53** शाखाको संस्करण तलको भागमा छ। कन्फिलक्ट समाधान गर्नका लागि, तपाईंले एक पक्ष छान्नु पर्नेछ वा दुवैको सामग्रीलाई आफै मर्ज गर्नु पर्नेछ। उदाहरणका लागि, तपाईंले यो कन्फिलक्ट यसरी समाधान गर्न सक्नुहुन्छ:

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

यसपछि, तपाईंले परिवर्तनहरूको समाधान गर्ने फाइललाई Gitमा थप्नु पर्छ र नयाँ कमिट बनाउन सक्नुहुन्छ:

```
$ git add index.html
$ git commit -m "Resolved merge conflict in index.html"
```

यसरी कन्फिलक्ट समाधान भएपछि, Git मर्ज प्रक्रिया पूरा हुनेछ र तपाईं आफ्नो काम अगाडि बढाउन सक्नुहुन्छ।

तपाईंले कन्फिलक्ट समाधान गर्दा प्रत्येक खण्डको थोरै हिस्सा राख्नुभयो, र <<<<<, =====, र >>>>> को लाइनहरू पूरै हटाइदिनुभएको छ। तपाईंले प्रत्येक कन्फिलक्ट भएका फाइलहरूमा यी खण्डहरू समाधान गरेपछि, तपाईंले git add चलाएर ती फाइलहरूलाई समाधान गरिएको भनेर चिन्हित गर्नुपर्नेछ। फाइललाई स्टेज गर्दा, Gitलाई थाहा हुन्छ कि यो फाइल समाधान भएको हो। यदि तपाईंले यी समस्याहरू समाधान गर्नको लागि ग्राफिकल उपकरण प्रयोग गर्न चाहनुहुन्छ भने, तपाईं git mergetool चलाउन सक्नुहुन्छ, जसले तपाईंलाई उपयुक्त भिज्युअल मर्ज टुल खोल्नमा मद्दत गर्दछ र कन्फिलक्टहरू समाधान गर्न मार्गदर्शन गर्दछ:

```
$ git mergetool
```

यदि तपाईंले git mergetool चलाउँदा यस्तो सन्देश देख्नुभयो भने:

```
This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuse diffmerge
ecmerge
p4merge araxis bc3 codecompare vimdiff emerge
```

यसको अर्थ हो कि तपाईंको मर्ज टुल कन्फिगर गरिएको छैन। तपाईंले चाहेको मर्ज टुलको चयन गर्नका लागि, यहाँ माथि उल्लेख गरिएको सूचीबाट एक टुल चयन गर्न सक्नुहुन्छ। उदाहरणको लागि, यदि तपाईं meld प्रयोग गर्न चाहनुहुन्छ भने, **तपाईं git mergetool --tool=meld** चलाउन सक्नुहुन्छ।

अर्को उदाहरणका रूपमा, यदि तपाईं opendiff प्रयोग गर्न चाहनुहुन्छ भने, निम्न आदेश प्रयोग गर्नुहोस्:

```
$ git mergetool --tool=opendiff
```

यसपछि, मर्ज टुलले कन्फिलक्टलाई दृश्य रूपमा देखाउँछ र तपाईंलाई कन्फिलक्ट समाधान गर्न मार्गदर्शन गर्दछ। जब तपाईंले कन्फिलक्ट समाधान गरेपछि फाइलहरू git add गरेर समाधानलाई चिन्हित गर्न सक्नुहुन्छ र अन्ततः git commit गरेर मर्ज प्रक्रिया समाप्त गर्न सक्नुहुन्छ:

```
$ git add index.html  
$ git commit -m "Resolved merge conflict in index.html"
```

यसरी, मर्ज कन्फिलक्टहरू समाधान गर्न ग्राफिकल उपकरणहरूको प्रयोग तपाईंलाई अझ सहज र दृश्यात्मक तरीका दिन्छ।

जबसम्म तपाईं मर्ज टुलबाट बाहिर जानुहुन्छ, Git तपाईंलाई सोध्छ कि मर्ज सफल भयो कि छैन। यदि तपाईं स्क्रिप्टलाई भन्नुहुन्छ कि यो सफल भयो भने, यसले तपाईंको फाइललाई स्टेज गरेर समाधानको रूपमा चिन्हित गर्दछ। त्यसपछि तपाईं git status फेरि चलाउन सक्नुहुन्छ यो पक्का गर्नेका लागि कि सबै कन्फिलक्टहरू समाधान गरिएका छन्:

```
$ git status  
On branch master  
All conflicts fixed but you are still merging.  
  (use "git commit" to conclude merge)  
Changes to be committed:  
  modified: index.html
```

यदि तपाईं यसबाट सन्तुष्ट हुनुहुन्छ र तपाईंले सुनिश्चित गर्नुभएको छ कि सबै कन्फिलक्ट भएका फाइलहरू स्टेज गरिएको छ, अब तपाईं git commit चलाएर मर्ज कमिटलाई अन्तिम रूप दिन सक्नुहुन्छ। डिफल्ट रूपमा, कमिट सन्देश यस प्रकारको देखिन्छ:

```
Merge branch 'iss53'  
Conflicts:  
  index.html  
#  
# It looks like you may be committing a merge.  
# If this is not correct, please remove the file  
# .git/MERGE_HEAD  
# and try again.  
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
# On branch master  
# All conflicts fixed but you are still merging.  
#  
# Changes to be committed:  
# modified: index.html  
#
```

यदि तपाईंलाई लाग्छ कि भविष्यमा अरूलाई मर्जका बारेमा बुझ्नमा मद्दत पुर्याउन सक्छ, तपाईं यस कमिट सन्देशलाई परिमार्जन गर्न सक्नुहुन्छ र यसले कसरी कन्फिलक्ट समाधान गरिएको छ र तपाईंले गरेका परिवर्तनहरूको कारण व्याख्या गर्न सक्नुहुन्छ यदि ती स्पष्ट छैन भने। यसरी तपाईंले कन्फिलक्टहरू समाधान गर्दा कमिट सन्देशलाई अझ स्पष्ट र सन्देशपूर्ण बनाउन सक्छौं।

जब तपाईं यसलाई स्वीकार गर्नुहुन्छ र कमिट सन्देश लेख्नुहुन्छ, मर्ज कमिट पूर्ण हुनेछ। ----- END -----