रिपोजिटरीमा परिवर्तनहरू रेकर्ड गर्दै

यस बिन्दुमा, तपाईंको स्थानीय मेसिनमा पूर्ण रूपमा कार्यात्मक Git रिपोजिटरी र यसको सबै फाइलहरूको वर्किङ प्रतिलिपि हुनु पर्छ। सामान्यतया, तपाईं परिवर्तनहरू गर्न सुरु गर्नुहुन्छ र जब प्रोजेक्टले रेकर्ड गर्न चाहिने अवस्था प्राप्त गर्छ, तब ती परिवर्तनहरूको snapshots commit गर्नुहुन्छ।

Git मा फाइलका अवस्थाहरू

तपाईंको वर्किङ डाइरेक्टरीमा प्रत्येक फाइल दुई अवस्थामध्ये एकमा हुन सक्छ: **tracked** वा **untracked**।

1. **Tracked Files**:
   यी फाइलहरू हुन्:
   - जुन अन्तिम commit (snapshot) मा समावेश थिए।
   - जुन commit को लागि staged गरिएका छन्।
     Tracked फाइलहरू तीन अवस्थाहरूमा हुन सक्छन्:
   - **Unmodified** (परिवर्तन नभएको)।
   - **Modified** (परिवर्तन गरिएको तर staged नभएको)।
   - **Staged** (commit गर्न तयार)।

सरल भाषामा भन्नुपर्दा, tracked फाइलहरू ती हुन् जसलाई Git ले चिन्छ।

2. **Untracked Files**:
   यी फाइलहरू हुन्:
   - जुन अन्तिम snapshot मा समावेश थिएनन्।
   - जसलाई staged गरिएको छैन।

Untracked फाइलहरू ती हुन् जसलाई Git ले चिनेको छैन।

Git को सामान्य Workflow

- जब तपाईंले पहिलो पटक एउटा रिपोजिटरी clone गर्नुहुन्छ, सबै फाइलहरू **tracked** र **unmodified** हुन्छन्, किनभने Git ले तिनलाई भर्खर checkout गरेको हुन्छ।
- जब तपाईं फाइलहरू edit गर्नुहुन्छ, Git ले तिनलाई **modified** रूपमा चिन्ह लगाउँछ किनभने ती अन्तिम commit भन्दा फरक छन्।
- edit गरेपछि, तपाईंले **stage** गरेर परिवर्तनहरू चयन गर्नुहुन्छ जुन तपाईं commit मा समावेश गर्न चाहनुहुन्छ।
- त्यसपछि, तपाईंले staged परिवर्तनहरू commit गरेर नयाँ snapshot सिर्जना गर्नुहुन्छ।
- यो edit, stage, र commit गर्ने चक्र तपाईंले प्रोजेक्टमा काम गर्दा दोहोरिन्छ।

फाइलहरूको स्थिति जाँच गर्दै

फाइलहरूको कुन अवस्था छ भनेर जाँच गर्नका लागि मुख्य उपकरण git status कमाण्ड हो। यदि तपाईं यो कमाण्ड clone गरेपछि तुरुन्तै चलाउनुहुन्छ भने, तपाईंले निम्न देखु हुनेछ:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

यसको अर्थ तपाईंको working directory सफा छ; अर्को शब्दमा, तपाईंको कुनै पनि tracked फाइलहरू परिवर्तन भएका छैनन्। Git ले कुनै पनि untracked फाइलहरू देख्दैन, अन्यथा ती यहाँ सूचीबद्ध हुने थिए। अन्त्यमा, कमाण्डले तपाईं कुन branch मा हुनुहुन्छ भनेर जानकारी दिन्छ र यो शाखा server मा उही शाखाबाट भिन्न छैन भनेर सूचित गर्दछ। अहिलेको लागि, यो शाखा सँधै master हो, जुन default हो; तपाईं यहाँ यसबारे चिन्ता गर्नुहुन्न। Git Branching ले शाखाहरू र references को बारेमा विस्तारमा बताउनेछ।

GitHub ले 2020 को मध्यतिर default branch नाम master बाट main मा परिवर्तन गरेको थियो, र अन्य Git hosts ले पनि यो परिवर्तनलाई अनुसरण गरे। त्यसैले तपाईंले केही नयाँ सिर्जना गरिएका repositories मा default branch नाम main पाउन सक्नुहुन्छ र master होइन। यसबाहेक, default branch नाम परिवर्तन गर्न सकिन्छ (जसरी तपाईंले आफ्नो default branch नाममा देख्नुभएको छ), त्यसैले तपाईंले default branch को लागि भिन्न नाम देख्न सक्नुहुन्छ।

यद्यपि, Git आफैंले अझै master लाई default को रूपमा प्रयोग गर्दछ, त्यसैले हामी यसलाई पुस्तकभरि प्रयोग गर्नेछौं।

मानौं तपाईं आफ्नो प्रोजेक्टमा एउटा नयाँ फाइल थप्नुहुन्छ, एउटा साधारण README फाइल। यदि फाइल पहिले अस्तित्वमा थिएन, र तपाईं git status चलाउनुहुन्छ भने, तपाईंले आफ्नो untracked फाइललाई यसरी देख्नुहुनेछ:

```
$ echo 'My Project' > README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    README
nothing added to commit but untracked files present (use "git add" to track)
```

तपाईंले देख्न सक्नुहुन्छ कि तपाईंको नयाँ README फाइल untracked छ, किनभने यो तपाईंको status आउटपुटको “Untracked files” शीर्षक अन्तर्गत छ। Untracked को अर्थ Git ले एउटा फाइल देखेको छ जुन तपाईंको अघिल्लो snapshot (commit) मा थिएन, र जुन अहिलेसम्म staged भएको छैन; Git ले यसलाई तपाईंको commit snapshots मा समावेश गर्न सुरु गर्दैन जबसम्म तपाईंले यो स्पष्ट रूपमा गर्न निर्देशन दिनुहुन्न। यसले तपाईंले गल्तीले binary files वा अन्य फाइलहरू समावेश गर्न सुरु नगर्नुहुन्छ। तपाईं README समावेश गर्न चाहनुहुन्छ, त्यसैले फाइल ट्र्याक गर्न सुरु गरौं।

नयाँ फाइलहरू ट्र्याक गर्दै

नयाँ फाइल ट्र्याक गर्न सुरु गर्नका लागि तपाईं git add कमाण्ड प्रयोग गर्नुहुन्छ। README फाइल ट्र्याक गर्न सुरु गर्न, तपाईं यो कमाण्ड चलाउन सक्नुहुन्छ:

```
$ git add README
```

यदि तपाईं फेरि git status कमाण्ड चलाउनुहुन्छ भने, तपाईंले देख्नुहुनेछ कि तपाईंको README फाइल अब ट्र्याक गरिएको छ र commit गर्न staged गरिएको छ:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  new file: README
```

तपाईंले बुझ्न सक्नुहुन्छ कि यो staged गरिएको छ किनभने यो "Changes to be committed" शीर्षक अन्तर्गत छ। यदि तपाईं यस बिन्दुमा commit गर्नुहुन्छ भने, तपाईंले git add चलाउँदाको समयमा फाइलको जुन संस्करण थियो, त्यो अर्को historical snapshot मा रहनेछ। तपाईंले सम्झन सक्नुहुन्छ कि जब तपाईंले पहिले git init चलाउनुभयो, त्यसपछि तपाईंले git add <files> चलाउनुभयो – त्यो तपाईंको directory का फाइलहरू ट्र्याक गर्न सुरु गर्नका लागि थियो।

git add कमाण्डले कुनै पनि फाइल वा directory को पथ लिन्छ; यदि यो directory हो भने, यो कमाण्डले recursive रूपमा directory भित्रका सबै फाइलहरूलाई थप्छ।

स्टेजिङ परिमार्जित फाइलहरू

तपाईंले पहिले नै ट्र्याक गरिएको फाइल (जस्तै, CONTRIBUTING.md) परिवर्तन गर्नुभयो भने र त्यसपछि git status कमाण्ड चलाउनुहुन्छ भने, तपाईंले यस्तो केही देख्नुहुनेछ:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  new file: README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
  modified: CONTRIBUTING.md
```

CONTRIBUTING.md फाइल “Changes not staged for commit” सेक्सनमा देखा पर्दछ – यसको अर्थ हो कि एक ट्र्याक गरिएको फाइल कार्य क्षेत्र (working directory) मा परिवर्तन भएको छ तर अझै स्टेज गरिएको छैन। यसलाई स्टेज गर्न, तपाईंले git add कमाण्ड चलाउनुहुन्छ। git add एक बहुउद्देश्यीय कमाण्ड हो – यो नयाँ फाइलहरू ट्र्याक गर्न, फाइलहरू स्टेज गर्न, र अन्य कामहरू जस्तै मर्ज-संघर्षित फाइलहरू समाधान भएको मार्क गर्न प्रयोग गरिन्छ। यो उपयोगी हुन सक्छ यसलाई “अर्को कमिटको लागि यो सामग्री सटीक रूपमा थप्नुहोस्” भनेर सोच्नुहोस् “यो फाइललाई प्रोजेक्टमा थप्नुहोस्” भन्दा। अब, CONTRIBUTING.md फाइललाई स्टेज गर्नका लागि git add चलाउँछौं, र फेरि git status चलाउँछौं:

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  new file: README
  modified: CONTRIBUTING.md
```

दुवै फाइलहरू स्टेज भएका छन् र तपाईंको अर्को कमिटमा जानेछन्। यस बिन्दुमा, मानौं तपाईंले CONTRIBUTING.md मा सानो परिवर्तन सम्झनुहुन्छ जुन तपाईं कमिट गर्नु अघि गर्न चाहनुहुन्छ। तपाईं यसलाई फेरि खोल्नुहुन्छ, परिवर्तन गर्नुहुन्छ, र कमिट गर्न तयार हुनुहुन्छ। तथापि, फेरि git status चलाउँछौं:

```
$ vim CONTRIBUTING.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
```

```
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  new file: README
  modified: CONTRIBUTING.md

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
  modified: CONTRIBUTING.md
```

के भयो? अब CONTRIBUTING.md “staged” र “unstaged” दुबै रूपमा सूचीबद्ध छ। यो सम्भव कसरी छ? यो थाहा हुन्छ कि Git ले फाइललाई ठीक त्यस अवस्थामा स्टेज गर्दछ जब तपाईं git add कमाण्ड चलाउनुहुन्छ। यदि तपाईं अहिले कमिट गर्नुहुन्छ भने, फाइलको संस्करण अन्तिम पटक तपाईंले git add चलाउनु भएको बेला जस्तै छ, त्यो संस्करण नै कमिटमा जानेछ, कार्यक्षेत्र (working directory) मा फाइलको हालको संस्करण होइन। यदि तपाईंले git add पछि फाइल परिवर्तन गर्नुभयो भने, तपाईंले फाइलको नवीनतम संस्करण स्टेज गर्न फेरि git add चलाउनुपर्छ:

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  new file: README
  modified: CONTRIBUTING.md
```

संक्षिप्त स्थिति (Short Status)

git status को आउटपुट धेरै विस्तृत हुन्छ तर लामो र शब्दजालले भरिएको हुन सक्छ। यसलाई सरल र छोटो बनाउन, git मा *short status flag* प्रयोग गर्न सकिन्छ। git status -s वा git status --short चलाउँदा कमाण्डको धेरै सरल आउटपुट प्राप्त हुन्छ:

```
$ git status -s
M   README
MM  Rakefile
A   lib/git.rb
M   lib/simplegit.rb
??  LICENSE.txt
```

यस आउटपुटलाई बुझ्नको लागि:

- ?? : नयाँ फाइलहरू जुन ट्र्याक भएका छैनन्।
- A : नयाँ फाइलहरू जुन स्टेज गरिएका छन्।
- M : परिमार्जित (modified) फाइलहरू।

दुई स्तम्भहरूको व्याख्या:

1. **पहिलो स्तम्भ**: स्टेजिङ क्षेत्रको स्थिति जनाउँछ।
2. **दोस्रो स्तम्भ**: कार्य क्षेत्र (working tree) को स्थिति जनाउँछ।

उदाहरण:

- M README: README फाइल कार्यक्षेत्रमा परिमार्जित छ तर अझै स्टेज गरिएको छैन।
- MM Rakefile: Rakefile फाइल पहिले परिमार्जित गरी स्टेज गरिएको थियो, त्यसपछि पुनः परिमार्जन गरिएको छ।
- A lib/git.rb: lib/git.rb नयाँ फाइल हो, जुन स्टेज गरिएको छ।
- M lib/simplegit.rb: lib/simplegit.rb फाइल परिमार्जित गरी स्टेज गरिएको छ।
- ?? LICENSE.txt: LICENSE.txt नयाँ फाइल हो, जुन अझै ट्र्याक गरिएको छैन।

फाइलहरू बेवास्ता गर्ने

धेरैजसो अवस्थामा, तपाईंलाई यस्तो प्रकारका फाइलहरू चाहिन्छ जुन Git ले स्वतः थप नगरोस् वा ट्र्याक नभएको भनेर देखाओस्। यी सामान्यतया स्वचालित रूपमा उत्पादन गरिएका फाइलहरू जस्तै log फाइलहरू वा build प्रणालीबाट उत्पादित फाइलहरू हुन सक्छन्। यस्ता अवस्थामा, तपाईंले .gitignore नामक एउटा फाइल सिर्जना गर्न सक्नुहुन्छ, जसमा ती फाइलहरूको pattern समावेश हुन्छ। यहाँ एउटा उदाहरण .gitignore फाइल छ:

```
$ cat .gitignore
*.[oa]
*~
```

पहिलो लाइनले Git लाई ".o" वा ".a" मा अन्त्य हुने कुनै पनि फाइल बेवास्ता गर्न भन्छ – जुन object र archive फाइलहरू हुन सक्छन्, जसले तपाईंको कोड बनाउनको क्रममा उत्पादन भएको हुन सक्छ। दोस्रो लाइनले Git लाई ती सबै फाइलहरू बेवास्ता गर्न भन्छ जसको नाम tilde (~) मा अन्त्य हुन्छ, जुन धेरै text editor जस्तै Emacs ले temporary फाइलको रूपमा प्रयोग गर्छ।

तपाईं log, tmp, वा pid directory; स्वचालित रूपमा उत्पादन गरिएका documentation; आदि पनि समावेश गर्न सक्नुहुन्छ। तपाईंको नयाँ repository को लागि .gitignore फाइल सुरुमै सेट अप गर्नु सामान्यतया राम्रो विचार हो, ताकि तपाईं गल्तीले पनि ती फाइलहरू commit नगर्नुहोस्, जुन तपाईं वास्तवमै आफ्नो Git repository मा राख्न चाहनुहुन्न।

तपाईंले .gitignore फाइलमा राख्न सक्ने pattern का लागि नियमहरू तल प्रस्तुत गरिएको छन्:

• खाली लाइनहरू वा # बाट सुरु हुने लाइनहरू बेवास्ता गरिन्छ।
• मानक glob pattern हरू काम गर्छन्, र यी सम्पूर्ण काम गर्ने वृक्षमा पुनरावृत्तिको रूपमा लागू गरिन्छ।
• तपाईं pattern हरूलाई forward slash (/) बाट सुरु गर्न सक्नुहुन्छ ताकि पुनरावृत्तिबाट बच्न सकियोस्।
• तपाईं pattern हरूलाई forward slash (/) बाट अन्त्य गर्न सक्नुहुन्छ जसले directory जनाउँछ।
• तपाईं pattern लाई नकार्नका लागि यसलाई exclamation mark (!) बाट सुरु गर्न सक्नुहुन्छ।

Glob pattern हरू भनेको simplified regular expressions जस्तै हुन्छन् जुन shells ले प्रयोग गर्छन्। एक asterisk (*) ले शून्य वा बढी क्यारेक्टरलाई मेल गर्छ; [abc] ले ब्राकेट भित्रका कुनै पनि क्यारेक्टर (यस अवस्थामा a, b, वा c) लाई मेल गर्छ; एक प्रश्न चिह्न (?) ले एकल क्यारेक्टरलाई मेल गर्छ; र हाइफन ([-]) बाट छुट्याएका क्यारेक्टरलाई समेटेका ब्राकेट (जस्तै [0-9]) ले ती क्यारेक्टर बीचका सबै क्यारेक्टरलाई मेल गर्छ (यस अवस्थामा 0 देखि 9 सम्म)। तपाईं nested directory हरूलाई मेल गर्न दुई asterisk पनि प्रयोग गर्न सक्नुहुन्छ; a/**/z ले a/z, a/b/z, a/b/c/z, र यस्तै अन्यलाई मेल गर्छ।

```
# ignore all .a files
*.a
# but do track lib.a, even though you're ignoring .a files above
!lib.a
# only ignore the TODO file in the current directory, not subdir/TODO
/TODO
# ignore all files in any directory named build
build/
```

```
# ignore doc/notes.txt, but not doc/server/arch.txt
doc/*.txt
# ignore all .pdf files in the doc/ directory and any of its subdirectories
doc/**/*.pdf
```

GitHub ले विभिन्न परियोजना र भाषाहरूको लागि रामो .gitignore फाइलका उदाहरणहरूको व्यापक सूची राखेको छ। तपाईं आफ्नो परियोजनाको लागि सुरूवात गर्न चाहनुहुन्छ भने, https://github.com/github/gitignore मा जान सक्नुहुन्छ।

सरल अवस्थामा, एउटा repository मा यसको root directory भित्र एउटा मात्र .gitignore फाइल हुन सक्छ, जसले सम्पूर्ण repository मा पुनरावृत्तिको रूपमा लागू हुन्छ। यद्यपि, subdirectory हरूमा थप .gitignore फाइलहरू पनि राख्न सकिन्छ। यी nested .gitignore फाइलका नियमहरू केवल त्यस फाइल रहेको directory अन्तर्गतका फाइलहरूमा मात्र लागू हन्छ।

तपाईंले गरिरहनुभएको Staged र Unstaged परिवर्तनहरू हेर्ने

यदि git status आदेश तपाईंका लागि धेरै साधारण छ र तपाईंलाई **के परिवर्तन गरियो भन्ने सटीक जानकारी** चाहिन्छ भने, तपाईं git diff आदेश प्रयोग गर्न सक्नुहुन्छ। यो आदेश तपाईंले सबैभन्दा धेरै यी दुई प्रश्नहरूको उत्तर दिन प्रयोग गर्नुहुनेछ:

1. तपाईंले के परिवर्तन गरिरहनुभएको छ तर अझै **staged** गरिएको छैन?
2. तपाईंले के **staged** गरिसक्नुभएको छ जुन commit मा जान्छ?

git status ले यी प्रश्नहरूलाई सामान्य रूपमा फाइलको नाम देखाएर उत्तर दिन्छ, तर git diff ले सटीक लाइनहरू देखाउँछ जुन थपिएका छन् वा हटाइएका छन् (patch)।

उदाहरण

मान्नुहोस् तपाईंले README फाइल फेरि सम्पादन गरेर **staged** गर्नुभयो, र CONTRIBUTING.md फाइल सम्पादन गर्नुभयो तर **staged** गर्नुभएको छैन।

git status आदेश चलाउँदा, यसरी देखिनेछ:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  modified: README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
  modified: CONTRIBUTING.md
```

Unstaged परिवर्तन हेर्न git diff आदेश

git diff आदेश बिना अन्य कुनै विकल्प चलाउँदा, यसले तपाईंको **working directory** मा भएका unstaged परिवर्तनहरू देखाउँछ।

```
$ git diff
diff --git a/CONTRIBUTING.md b/CONTRIBUTING.md
index 8ebb991..643e24f 100644
--- a/CONTRIBUTING.md
+++ b/CONTRIBUTING.md
@@ -65,7 +65,8 @@ branch directly, things can get messy.
Please include a nice description of your changes when you submit your PR;
if we have to read the whole diff to figure out why you're contributing
in the first place, you're less likely to get feedback and have your change
-merged in.
+merged in. Also, split your changes into comprehensive chunks if your patch is
+longer than a dozen lines.
```

Staged परिवर्तन हेर्न git diff --staged आदेश

git diff --staged (वा --cached) आदेश प्रयोग गर्दा, यो तपाईंले staged गरेका परिवर्तनहरूलाई अघिल्लो commit सँग तुलना गरेर देखाउँछ।

```
$ git diff --staged
diff --git a/README b/README
new file mode 100644
index 0000000..03902a1
--- /dev/null
+++ b/README
@@ -0,0 +1 @@
+My Project
```

दुवै Staged र Unstaged परिवर्तनहरू हेर्न

मान्नुहोस् तपाईंले CONTRIBUTING.md फाइल staged गर्नुभयो र त्यसपछि फेरि सम्पादन गर्नुभयो।

```
$ git add CONTRIBUTING.md
$ echo '# test line' >> CONTRIBUTING.md
$ git status
```

अब git diff ले unstaged परिवर्तन देखाउँछ:

```
$ git diff
+# test line
```

र git diff --cached ले staged परिवर्तन देखाउँछ:

```
$ git diff --cached
@@ -65,7 +65,8 @@ merged in.
+longer than a dozen lines.
```

git difftool आदेश प्रयोग

यदि तपाईं **graphical** वा बाह्य software (जस्तै vimdiff, emerge) प्रयोग गरेर diffs हेर्न चाहनुहुन्छ भने, git difftool आदेश प्रयोग गर्न सक्नुहुन्छ।
उपलब्ध tools हेर्नका लागि:

```
$ git difftool --tool-help
```

तपाईंको परिवर्तनहरू Commit गर्दै

अब तपाईंको staging area इच्छाअनुसार सेटअप भएको छ भने, तपाईं आफ्नो परिवर्तनहरू commit गर्न सक्नुहुन्छ। याद गर्नुहोस् कि कुनै पनि unstaged फाइलहरू – तपाईंले **git add** चलाएको छैन भने ती फाइलहरू यो commit मा समावेश हुने छैनन्। ती फाइलहरू तपाईंको डिस्कमा modified फाइलकै रूपमा रहनेछन्।

उदाहरणका लागि, मानौं कि तपाईंले अन्तिम पटक **git status** चलाउँदा सबै फाइलहरू staged देख्नु भयो, र तपाईं आफ्नो परिवर्तन commit गर्न तयार हुनुहुन्छ। Commit गर्न सबैभन्दा सरल तरिका भनेको निम्न कमाण्ड प्रयोग गर्नु हो:

```
$ git commit
```

यसले तपाईंको प्राथमिक editor (जस्तै, vim वा nano) खोल्छ।

यो तपाईंको shell को **EDITOR** environment variable द्वारा सेट गरिएको हुन्छ – सामान्यतः vim वा emacs। यद्यपि, तपाईं यसलाई आफ्नो इच्छाअनुसार निम्न कमाण्ड प्रयोग गरेर परिवर्तन गर्न सक्नुहुन्छ:

तपाईंले के-कसरी परिमार्जन गर्नुभएको छ भन्ने अझ स्पष्ट सम्झनाका लागि, तपाईं git commit मा -v अप्शन पास गर्न सक्नुहुन्छ। यसले तपाईंको परिमार्जनको डिफ (diff) सम्पादकमा राख्छ, जसले तपाईंलाई ठ्याक्कै के परिवर्तनहरू कमिट गर्दै हुनुहुन्छ भन्ने हेर्न अनुमति दिन्छ।

```
# On branch master
# तपाईँको शाखा 'origin/master' संग अप-टु-डेट छ।
#
# Commit गर्नका लागि परिवर्तनहरू:
# नयाँ फाइल: README
# परिमार्जित: CONTRIBUTING.md
#
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

तपाईंले देख्न सक्नुहुन्छ कि डिफल्ट कमिट सन्देशमा git status कमाण्डको पछिल्लो आउटपुट कमेन्ट गरिएको र माथि एक खाली लाइन समावेश छ। यी कमेन्टहरू हटाएर आफ्नो कमिट सन्देश लेख्न सक्नुहुन्छ, अथवा तपाईंलाई के कमिट गर्दै हुनुहुन्छ भनेर सम्झन मद्दत गर्न तिनीहरू त्यहाँ राख्न पनि सक्नुहुन्छ।

जब तपाईं सम्पादकबाट बाहिर निस्किनुहुन्छ, Git ले त्यो कमिट सन्देशको साथमा (कमेण्टहरू र डिफ हटाएर) तपाईंको कमिट तयार गर्छ। वैकल्पिक रूपमा, तपाईं कमिट कमाण्डमा -m फ्ल्याग प्रयोग गरी इनलाइनमा आफ्नो कमिट सन्देश टाइप गर्न सक्नुहुन्छ, जस्तै:

```
$ git commit -m "Story 182: fix benchmarks for speed"
[master 463dc4f] Story 182: fix benchmarks for speed
2 files changed, 2 insertions(+)
create mode 100644 README
```

अब तपाईंले आफ्नो पहिलो कमिट बनाइसक्नुभएको छ! तपाईंले कमिटले केही जानकारी दिएको देख्न सक्नुहुन्छ: कुन ब्रान्चमा तपाईंले कमिट गर्नुभयो (master), कमिटको SHA-1 चेकसम (463dc4f), कति फाइलहरू परिवर्तन भए, र कमिटमा थपिएका र हटाइएका लाइनहरूको तथ्यांक।

याद गर्नुहोस् कि कमिटले तपाईंको स्टेजिङ एरियामा सेट गरिएको स्न्यापशट रेकर्ड गर्छ। जुनसुकै कुरा जुन तपाईंले स्टेज गर्नुभएको छैन, अझै परिमार्जित अवस्थामा रहन्छ; तपाईं अर्को कमिट गर्न सक्नुहुन्छ र यसलाई आफ्नो इतिहासमा थप्न सक्नुहुन्छ।
हरेक पटक तपाईं कमिट गर्नुहुन्छ, तपाईं आफ्नो प्रोजेक्टको एउटा स्न्यापशट रेकर्ड गर्नुहुन्छ, जसलाई पछि फर्काउन वा तुलना गर्न प्रयोग गर्न सकिन्छ।

स्टेजिङ एरिया स्किप गर्ने

स्टेजिङ एरियाले तपाईंलाई कमिटहरू आफ्नो इच्छाअनुसार बनाउन धेरै उपयोगी हुन सक्छ, तर कहिलेकाहीं यो तपाईंको वर्कफ्लोमा आवश्यकताभन्दा बढी जटिल हुन सक्छ। यदि तपाईं स्टेजिङ एरियालाई स्किप गर्न चाहनुहुन्छ भने, Git ले एक सरल शर्टकट प्रदान गर्छ। `git commit` कमाण्डमा `-a` अप्शन थप्दा, Git ले कमिट गर्नुअघि पहिल्यै ट्र्याक गरिएको सबै फाइलहरू स्वतः स्टेज गर्छ, जसले तपाईंलाई git add भाग स्किप गर्न अनुमति दिन्छ:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
  modified: CONTRIBUTING.md
no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -a -m 'Add new benchmarks'
[master 83e38c7] Add new benchmarks
1 file changed, 5 insertions(+), 0 deletions(-)
```

ध्यान दिनुहोस् कि यस अवस्थामा, `CONTRIBUTING.md` फाइलमा `git add` चलाउन तपाईंलाई आवश्यक पर्दैन। त्यसको कारण `-a` फ्ल्यागले सबै परिवर्तन भएका फाइलहरू समावेश गर्दछ। यो सुविधाजनक छ, तर सावधान रहनुहोस्; कहिलेकाहीं यस फ्ल्यागले अनावश्यक परिवर्तनहरू पनि समावेश गर्न सक्छ।

Git बाट फाइलहरू हटाउने

Git बाट कुनै फाइल हटाउन, तपाईंले त्यसलाई ट्र्याक गरिएका फाइलहरूबाट हटाउनुपर्छ (अर्थात्, स्टेजिङ एरियाबाट हटाउनुपर्छ) र त्यसपछि कमिट गर्नुपर्छ। git rm कमाण्डले यो काम गर्छ र त्यससाथै फाइललाई तपाईंको वर्किङ डाइरेक्टरीबाट पनि हटाउँछ ताकि अर्को पटक git status चलाउँदा त्यो फाइल अनट्र्याक गरिएको फाइलको रूपमा देखिने छैन।

यदि तपाईंले वर्किङ डाइरेक्टरीबाट मात्र फाइल हटाउनुभयो भने, त्यो फाइल Changes not staged for commit (अनस्टेज) क्षेत्रमा git status को आउटपुटमा देखिन्छ:

```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
  deleted: PROJECTS.md
no changes added to commit (use "git add" and/or "git commit -a")
```

यदि तपाईं git rm चलाउनुहुन्छ भने, यसले फाइललाई हटाउन स्टेज गर्छ:

```
$ git rm PROJECTS.md
rm 'PROJECTS.md'
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
  deleted: PROJECTS.md
```

अर्को पटक तपाईंले कमिट गर्दा, फाइल हटिसकेको र ट्र्याक नगरिएको हुनेछ।
यदि तपाईंले फाइललाई परिवर्तन गर्नुभएको थियो वा पहिले नै स्टेजिङ एरियामा थप्नुभएको थियो भने, `-f` अप्शन प्रयोग गरेर फोर्स हटाउनुपर्छ।
यो सुरक्षा सुविधा हो जसले कुनै स्न्यापसटमा रेकर्ड नगरिएको र Git बाट पुन: प्राप्त गर्न नसकिने डेटा दुर्घटनावश हट्न नदिन रोक्छ।

Git बाट फाइललाई वर्किङ ट्रीमा राख्दै स्टेजिङ एरियाबाट हटाउने

तपाईंले कहिलेकाहीँ फाइललाई आफ्नो वर्किङ ट्रीमा राख्न तर Git बाट ट्र्याक नगर्न चाहनुहुन्छ। यसको मतलब, तपाईं फाइललाई आफ्नो हार्ड ड्राइभमा राख्न चाहनुहुन्छ तर Git ले यसलाई ट्र्याक गर्न रोक्न चाहनुहुन्छ। यो विशेष गरी उपयोगी हुन्छ यदि तपाईंले आफ्नो `.gitignore` फाइलमा केही थप्न बिर्सनुभएको छ र दुर्घटनावश कुनै ठूलो लग फाइल वा कम्पाइल गरिएका `.a` फाइलहरू स्टेज गर्नुभयो भने।

यसका लागि, `--cached` अप्शन प्रयोग गर्नुस्:

```
$ git rm --cached README
```

git rm कमाण्डलाई तपाईं फाइलहरू, डाइरेक्टोरीहरू, र फाइल-ग्लोब ढाँचा दिन सक्नुहुन्छ।
यसको मतलब तपाईंले यस्ता कामहरू गर्न सक्नुहुन्छ:

```
$ git rm log/\*.log
```

नोट: * अघि स्ल्यास `(\)` राख्न आवश्यक छ किनभने `Git` ले फाइलनाम विस्तार आफ्नो हिसाबले गर्छ, तपाईंको शेलको फाइलनाम विस्तार बाहेक।
यो कमाण्डले `log/` डाइरेक्टोरीमा रहेका सबै `.log` एक्स्टेन्शन भएका फाइलहरू हटाउँछ।

त्यस्तै, तपाईं यस्तो पनि गर्न सक्नुहुन्छ:

```
$ git rm \*~
```

यो कमाण्डले सबै फाइलहरू हटाउँछ जसको नाम `~` मा समाप्त हुन्छ।

फाइल सार्न

धेरैजसो अन्य VCS जस्तो होइन, Git ले फाइलको मूभमेन्टलाई स्पष्ट रूपमा ट्र्याक गर्दैन। यदि तपाईंले Git मा फाइलको नाम परिवर्तन गर्नुभयो भने, Git मा कुनै पनि मेटाडाटा स्टोर हुँदैन जसले यो भन्छ कि तपाईंले फाइलको नाम परिवर्तन गर्नुभयो।
तर Git धेरै स्मार्ट छ र पछि यो पत्ता लगाउन सक्छ। Git मा फाइलको नाम परिवर्तन गर्न, तपाईं यस्तो कमाण्ड चलाउन सक्नुहुन्छ:

```
$ git mv file_from file_to
```

यो चल्छ। उदाहरणका लागि:

```
$ git mv README.md README
$ git status
```

Git ले यसलाई रिनेम गरिएको फाइलको रूपमा देख्छ:

```
renamed: README.md -> README
```

तर, यो तीन कमाण्डको बराबर हो:

```
$ mv README.md README
$ git rm README.md
$ git add README
```

git mv प्रयोग गर्नु एउटा कमाण्डमा काम गर्ने सजिलो तरिका हो।

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