कमिट इतिहास हेर्दै

तपाईले धेरै कमिटहरू बनाएपछि वा कुनै रेपो क्लोन गरेपछि जसमा पहिलेदेखि नै कमिट इतिहास छ, तपाईलाई के भएको थियो भनेर फर्केर हेर्न मन लाग्न सक्छ। यसका लागि सबैभन्दा आधारभूत र शक्तिशाली टुल git log कमाण्ड हो। यी उदाहरणहरू “simplegit” नामको एउटा सरल प्रोजेक्टमा आधारित छन्। यो प्रोजेक्ट प्राप्त गर्न, तलको कमाण्ड चलाउनुहोस्:

```
$ git clone https://github.com/schacon/simplegit-progit
```

जब तपाईं यो प्रोजेक्टमा git log चलाउनुहुन्छ, तपाईंलाई केहि यस प्रकारको आउटपुट देखिन सक्छः

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

डिफल्ट रूपमा, कुनै पनि आर्गुमेन्ट बिना, git log ले रिपोजिटरीमा भएका कमिटहरू उल्टो कालक्रममा देखाउँछ; अर्थात्, सबैभन्दा नयाँ कमिटहरू पहिलोमा देखिन्छन्। जस्तै तपाईंले देख्नुभयो, यस कमाण्डले प्रत्येक कमिटको SHA-1 चेकसम, लेखकको नाम र इमेल, लेखिएको मिति, र कमिट म्यासेज सूचीबद्ध गर्छ।

`git log` कमाण्डको धेरै विकल्पहरू उपलब्ध छन् जसले तपाईंलाई चाहिएको कुरा ठीक रूपमा देखाउँछ। यहाँ केही लोकप्रिय विकल्पहरू उल्लेख गरिएका छन्।

- एउटा उपयोगी विकल्प हो `-p` वा `--patch`, जसले प्रत्येक कमिटमा गरिएको परिवर्तन (प्याच आउटपुट) देखाउँछ।
- तपाईंले प्रदर्शित हुने लग प्रविष्टिहरूको संख्या पनि सीमित गर्न सक्नुहुन्छ, जस्तै, `-2` प्रयोग गरेर पछिल्ला दुई मात्र प्रविष्टिहरू हेर्न सकिन्छ।

```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700
  Change version number
diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,7 +5,7 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
   s.platform = Gem::Platform::RUBY
   s.name = "simplegit"
- s.version = "0.1.0"
+ s.version = "0.1.1"
   s.author = "Scott Chacon"
   s.email = "schacon@gee-mail.com"
   s.summary = "A simple gem for using Git in Ruby code."
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date: Sat Mar 15 16:40:33 2008 -0700
```

```
  Remove unnecessary test
diff --git a/lib/simplegit.rb b/lib/simplegit.rb
index a0a60ae..47c6340 100644
--- a/lib/simplegit.rb
+++ b/lib/simplegit.rb
@@ -18,8 +18,3 @@ class SimpleGit
   end
 end
-
-if $0 == __FILE__
- git = SimpleGit.new
- puts git.show
-end
```

यस विकल्पले उही जानकारी प्रदर्शन गर्छ तर प्रत्येक प्रविष्टिपछि सिधै डिफ (`diff`) देखाउँछ। यो कोड समीक्षाका लागि वा कुनै सहकर्मीले थपेका कमिटहरूको क्रममा के भएको छ भन्ने छिट्टै हेर्नका लागि धेरै उपयोगी हुन्छ। तपाईं git log मा विभिन्न संक्षिप्त विकल्पहरू पनि प्रयोग गर्न सक्नुहुन्छ। उदाहरणका लागि, यदि तपाईं प्रत्येक कमिटको छोटो संख्यात्मक तथ्यांकहरू हेर्न चाहनुहुन्छ भने, तपाईं `--stat` विकल्प प्रयोग गर्न सक्नुहुन्छ।

```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700
  Change version number
 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
```

```
Date: Sat Mar 15 16:40:33 2008 -0700

  Remove unnecessary test

 lib/simplegit.rb | 5 -----

 1 file changed, 5 deletions(-)

commit a11bef06a3f659402fe7563abf99ad00de2209e6

Author: Scott Chacon <schacon@gee-mail.com>

Date: Sat Mar 15 10:31:28 2008 -0700

  Initial commit

 README | 6 ++++++

 Rakefile | 23 +++++++++++++++++++++++

 lib/simplegit.rb | 25 +++++++++++++++++++++++++

 3 files changed, 54 insertions(+)
```

जसरी तपाईंले देख्न सक्नुहुन्छ, `--stat` विकल्पले प्रत्येक कमिट प्रविष्टिको तल परिमार्जित फाइलहरूको सूची, परिमार्जित फाइलहरूको संख्या, र ती फाइलहरूमा थपिएका तथा हटाइएका लाइनहरूको संख्या प्रिन्ट गर्छ। यसले जानकारीको सारांश पनि अन्त्यमा देखाउँछ। अर्को धेरै उपयोगी विकल्प --pretty हो। यो विकल्पले लग (log) आउटपुटलाई पूर्वनिर्धारित ढाँचाहरूभन्दा अन्य ढाँचामा परिवर्तन गर्छ। तपाईंले प्रयोग गर्न सक्ने केही पूर्वनिर्मित मानहरू उपलब्ध छन्। यसमा oneline मानले प्रत्येक कमिटलाई एकै लाइनमा प्रिन्ट गर्छ, जुन धेरै कमिटहरू हेर्दा उपयोगी हुन्छ। त्यसैगरी, `short`, `full`, र `fuller` मानहरूले कम जानकारी वा बढी जानकारीसहित लगभग उस्तै ढाँचामा आउटपुट देखाउँछन्।

```
$ git log --pretty=oneline

ca82a6dff817ec66f44342007202690a93763949 Change version number

085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 Remove unnecessary test

a11bef06a3f659402fe7563abf99ad00de2209e6 Initial commit
```

सबैभन्दा रोचक विकल्प मान `format` हो, जसले तपाईंलाई आफ्नो लग (`log`) आउटपुटको ढाँचा निर्दिष्ट गर्न अनुमति दिन्छ। यो विशेष गरी मेशिनद्वारा पार्सिङ (`machine parsing`) का लागि आउटपुट उत्पादन

गर्दा उपयोगी हुन्छ – किनभने तपाईंले ढाँचालाई स्पष्ट रूपमा निर्दिष्ट गर्न सक्नुहुन्छ, र यो Git को अपडेटसँग परिवर्तन हुँदैन।

```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : Change version number
085bb3b - Scott Chacon, 6 years ago : Remove unnecessary test
a11bef0 - Scott Chacon, 6 years ago : Initial commit
```

---

git log --pretty=format मा प्रयोग गर्न सकिने उपयोगी स्पेसिफायरहरूका सूचीहरूले format ले लिन सक्ने केही महत्वपूर्ण स्पेसिफायरहरू देखाउँछ।

स्पेसिफायर (specifier) र यसको अर्थ:

स्पेसिफायर	अर्थ
%H	कमिट ह्यास
%h	छोटकरी कमिट ह्यास
%T	ट्रि ह्यास
%t	छोटकरी ट्रि ह्यास
%P	अभिभावक ह्यासहरू
%p	छोटकरी अभिभावक ह्यासहरू
%an	लेखकको नाम
%ae	लेखकको इमेल
%ad	लेखकको मिति (--date= विकल्प अनुसारको ढाँचा)
%ar	लेखकको मिति, सापेक्ष
%cn	कमिटरको नाम
%ce	कमिटरको इमेल
%cd	कमिटरको मिति
%cr	कमिटरको मिति, सापेक्ष
%s	विषय (Subject)

Specifier	Description	Example
`%H`	Commit hash	`f8b8dbf5d5b658b4d8f09f8e1cfcc70fd87c4500`
`%h`	Abbreviated commit hash	`f8b8dbf`
`%T`	Tree hash	`89dbb7d52f57fddfd57b8f926272d59298f69d2b`
`%t`	Abbreviated tree hash	`89dbb7d`
`%P`	Parent hashes	`ac74b45 f9b0b44`
`%p`	Abbreviated parent hashes	`ac74b45 f9b0b44`
`%an`	Author name	`John Doe`
`%ae`	Author email	`john@example.com`
`%ad`	Author date (format respects the --date=option)	`Mon, 12 Jan 2024 14:42:00 +0000`
`%ar`	Author date, relative	`2 weeks ago`
`%cn`	Committer name	`Alice Smith`
`%ce`	Committer email	`alice@example.com`
`%cd`	Committer date	`Mon, 12 Jan 2024 15:10:00 +0000`
`%cr`	Committer date, relative	`2 weeks ago`
`%s`	Subject (commit message)	`Fixed issue with user login`

तपाईंलाई लेखक र कमिटर बीचको फरक के हो भन्ने जिज्ञासा हुन सक्छ। लेखक भनेको त्यो व्यक्ति हो जसले मूल रूपमा काम लेखेको हो, जबकि कमिटर भनेको त्यो व्यक्ति हो जसले अन्तिम पटक कामलाई लागू गरेको हो। उदाहरणका लागि, यदि तपाईंले एउटा प्याच (patch) प्रोजेक्टमा पठाउनुभयो र कुनै कोर सदस्यले त्यो प्याच लागू गर्यो भने, तपाईंलाई लेखकको रूपमा क्रेडिट मिल्छ र कोर सदस्यलाई कमिटरको रूपमा क्रेडिट मिल्छ। हामी यो भेदबुझाइलाई *डिस्ट्रिब्युटेड गिटमा* थप कुरा गर्नेछौं।

`oneline` र `format` विकल्पहरू विशेष रूपमा अर्को एक कमाण्ड विकल्प `--graph` सँग प्रयोग गर्दा उपयोगी हुन्छ। यस विकल्पले तपाईंको शाखा र मर्ज इतिहासको ASCII ग्राफ देखाउँछ:

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 Ignore errors from SIGCHLD on trap
* 5e3ee11 Merge branch 'master' of
https://github.com/dustin/grit.git
|\
| * 420eac9 Add method for getting the current branch
* | 30e367c Timeout code and tests
* | 5a09431 Add timeout protection to grit
* | e1193f8 Support for heads with slashes in them
|/
* d6016bc Require time for xmlschema
* 11d191e Merge branch 'defunkt' into local
```

यो प्रकारको आउटपुट अर्को अध्यायमा शाखा र मर्ज गर्दा अझ रोचक हुनेछ। यी सबै केबल `git log` को लागि केहि साधारण आउटपुट-फर्म्याटिङ विकल्पहरू हुन् – धेरै अन्य छन्। git log का सामान्य विकल्पहरू जसलाई हामीले हालसम्म कभर गरेका छौं, साथै केही अन्य सामान्य फर्म्याटिङ विकल्पहरू जसले log कमाण्डको आउटपुटलाई कसरी परिवर्तन गर्छ भन्ने कुरा तल सूचीबद्ध गरिएको छ।

विकल्प	विवरण
-p	प्रत्येक कमिटसँग प्रस्तुत गरिएको प्याच देखाउनुहोस्।
--stat	प्रत्येक कमिटमा संशोधित गरिएको फाइलहरूको लागि आँकडा देखाउनुहोस्।
--shortstat	--stat कमाण्डबाट मात्र परिवर्तन/संक्षिप्त/हटाइएका पंक्तिहरू देखाउनुहोस्।
--name-only	कमिट जानकारी पछि मात्र संशोधित गरिएको फाइलहरूको सूची देखाउनुहोस्।
--name-status	फाइलहरूको सूची देखाउनुहोस् जसमा थपिएको/परिवर्तित/हटाइएको जानकारी पनि हुनेछ।
--abbrev-commit	SHA-1 चेकसमको केवल पहिलो केही वर्णहरू देखाउनुहोस्, 40 का सट्टा।
--relative-date	पूर्ण मिति फारमेटको सट्टा, मिति सापेक्ष फारमेटमा (जस्तै, "२ हप्ता अगाडि") देखाउनुहोस्।
--graph	कमिट आउटपुटको सँगै शाखा र मर्ज इतिहासको ASCII ग्राफ देखाउनुहोस्।
--pretty	कमिटहरू वैकल्पिक फारमेटमा देखाउनुहोस्। विकल्प मानहरूमा oneline, short, full, fuller, र format (जहाँ तपाईं आफैंको फारमेट चयन गर्नुहुन्छ) समावेश छन्।
--oneline	--pretty=oneline --abbrev-commit को संक्षिप्त रूप प्रयोग गर्नुहोस्।

Option	Description	Example
`-p`	Show the patch introduced with each commit.	Displays the changes (diffs) for each commit.
`--stat`	Show statistics for files modified in each commit.	Displays the number of files changed, insertions, and deletions.
`--shortstat`	Display only the changed/insertions/deletions line from the `--stat` command.	Displays a summary of how many files were changed, inserted, and deleted.
`--name-only`	Show the list of files modified after the commit information.	Displays only the names of modified files.
`--name-status`	Show the list of files affected with added/modified/deleted information as well.	Displays the status (`A` , `M` , `D`) and names of modified files.
`--abbrev-commit`	Show only the first few characters of the SHA-1 checksum instead of all 40.	Displays a shortened commit hash (default 7 characters).
`--relative-date`	Display the date in a relative format (e.g., "2 weeks ago") instead of using the full date format.	Displays a relative date like "2 days ago".
`--graph`	Display an ASCII graph of the branch and merge history beside the log output.	Displays an ASCII graph to show the commit history and branches.
`--pretty`	Show commits in an alternate format. Option values include `oneline` , `short` , `full` , `fuller` , and `format` .	Customizes the output format (e.g., `--pretty=oneline` displays each commit in one line).
`--oneline`	Shorthand for `--pretty=oneline --abbrev-commit` used together.	Displays each commit in a single line with an abbreviated commit hash.

लिमिटिङ लग आउटपुट

आउटपुट-फर्म्याटिङ विकल्पहरूका अतिरिक्त, git log ले केही उपयोगी लिमिटिङ विकल्पहरू पनि स्वीकार गर्दछ; जसले तपाईंलाई केवल कमिटहरूको एक उपसमूह देखाउन अनुमति दिन्छ। तपाईंले एक यस्तो विकल्प पहिले नै देख्नु भएको छ - -2 विकल्प, जसले केवल अन्तिम दुई कमिटहरू देखाउँछ। वास्तवमा, तपाईं -<n> प्रयोग गर्न सक्नुहुन्छ, जहाँ n कुनै पनि पूर्णांक हो जसले अन्तिम n कमिटहरू देखाउँछ। व्यावहारिक रूपमा, तपाईं यो धेरै प्रयोग नगर्नुहोस्, किनकि Git ले डिफल्ट रूपमा सबै आउटपुटलाई पेजरमा पाइप गर्दछ, जसले गर्दा तपाईंलाई एक पटकमा केवल एक पृष्ठको लग आउटपुट देख्न अनुमति दिन्छ।

तर, समय-सीमा लिमिटिङ विकल्पहरू जस्तै `--since` र `--until` धेरै उपयोगी छन्। उदाहरणको रूपमा, यो कमाण्डले अन्तिम दुई हप्तामा गरिएका कमिटहरूको सूची प्राप्त गर्छ:

```
$ git log --since="2 weeks ago"
```

यसले तपाईंलाई दुई हप्ताभित्र भएका सबै कमिटहरू देखाउँछ।

तपाईं विशिष्ट मिति जस्तै "2008-01-15" वा सापेक्ष मिति जस्तै "2 वर्ष 1 दिन 3 मिनेट पहिले" निर्दिष्ट गर्न सक्नुहुन्छ।

उदाहरणको रूपमा:

1. विशिष्ट मिति प्रयोग गरेर:

   ```
   $ git log --since="2008-01-15"
   ```

2. सापेक्ष मिति प्रयोग गरेर:

   ```
   $ git log --since="2 years 1 day 3 minutes ago"
   ```

यसले तपाईंलाई सो मितिबाट वा सो सापेक्ष समयमा भएका सबै कमिटहरू देखाउँछ।

तपाईं सूचीलाई केही खोज मापदण्डसँग मिल्दो कमिटहरूमा फिल्टर गर्न सक्नुहुन्छ।

- **--author** विकल्पले तपाईंलाई विशेष लेखकमा आधारित कमिटहरू फिल्टर गर्न अनुमति दिन्छ।
- **--grep** विकल्पले तपाईंलाई कमिट सन्देशहरूमा कुञ्जीशब्दहरू खोज्न अनुमति दिन्छ।

उदाहरणका रूपमा:

1. विशेष लेखकका कमिटहरू देख्न:

   ```
   $ git log --author="Scott Chacon"
   ```

2. कमिट सन्देशमा विशिष्ट कुञ्जी शब्द खोज्न:

```
$ git log --grep="bug fix"
```

यी विकल्पहरू तपाईंलाई एकदम विशेष खोजहरू गर्न र तपाईंलाई चाहिने कमिटहरू छान्न मद्दत गर्छ।

तपाईं **--author** र **--grep** खोज मापदण्डहरूको एक भन्दा बढी उदाहरण निर्दिष्ट गर्न सक्नुहुन्छ, जसले कमिट आउटपुटलाई ती कमिटहरूमा सीमित गर्नेछ जुन कुनै पनि **--author** ढाँचासँग मेल खान्छ र कुनै पनि **--grep** ढाँचासँग मेल खान्छ। तर, यदि तपाईं **--all-match** विकल्प थप्नुहुन्छ भने, यसले आउटपुटलाई ती कमिटहरूमा मात्र सीमित गर्नेछ जुन सबै **--grep** ढाँचासँग मेल खान्छ।

उदाहरणहरू:

1. **--author** र **--grep** को एक भन्दा बढी ढाँचासँग मेल खाने कमिटहरू देख्न:

```
$ git log --author="Scott Chacon" --grep="bug fix"
```

2. **--all-match** विकल्पको साथ, सबै **--grep** ढाँचासँग मेल खाने कमिटहरू मात्र देख्न:

```
$ git log --author="Scott Chacon" --grep="bug fix" --all-match
```

यसरी, तपाईं थप सटीक रूपमा कमिटहरू खोज्न र फिल्टर गर्न सक्नुहुन्छ।

अर्को एक उपयोगी फिल्टर भनेको **-S** विकल्प हो, जसलाई सामान्यतया Git को "पिकएक्स" विकल्प भनेर चिनिन्छ। यसले एउटा स्ट्रिङ लिन्छ र केवल ती कमिटहरू देखाउँछ जसले त्यो स्ट्रिङको आवृत्तिको संख्या परिवर्तन गरेको छ। उदाहरणको लागि, यदि तपाईं कुनै विशेष फंक्शनको सन्दर्भ थपिएको वा हटाइएको अन्तिम कमिट खोज्न चाहनुहुन्छ भने, तपाईं यसरी चलाउन सक्नुहुन्छ:

```
$ git log -S function_name
```

अन्तिम उपयोगी विकल्प भनेको **path** हो। यदि तपाईं एउटा डाइरेक्टरी वा फाइल नाम निर्दिष्ट गर्नुहुन्छ भने, यसले केवल ती कमिटहरू देखाउँछ जसले ती फाइलहरूमा परिवर्तन ल्याएको छ। यो सधैं अन्तिम विकल्प हुन्छ र सामान्यतया विकल्पहरू र पथहरू बीचमा डबल ड्यासेस (--) द्वारा पृथक गरिन्छ:

```
$ git log -- path/to/file
```

यसरी, तपाईं सधैं एक विशिष्ट फाइल वा डाइरेक्टरीमा भएका परिवर्तनहरूमा ध्यान केन्द्रित गर्न सक्नुहुन्छ।

---

git log को कमिट आउटपुटलाई फिल्टर गर्नका लागि उपयोगी विकल्पहरू

यहाँ केहि सामान्य विकल्पहरूको तालिका दिइएको छ जसले git log को आउटपुटलाई सीमित गर्नमा मद्दत गर्दछ:

विकल्प	विवरण
-<n>	केवल पछिल्ला n कमिटहरू देखाउनुहोस्।
--since, --after	निर्दिष्ट मितिको पछि भएका कमिटहरू मात्र देखाउनुहोस्।
--until, --before	निर्दिष्ट मितिको अघि भएका कमिटहरू मात्र देखाउनुहोस्।
--author	केवल ती कमिटहरू देखाउनुहोस् जसको लेखक जानकारी निर्दिष्ट स्ट्रिङसँग मेल खान्छ।
--committer	केवल ती कमिटहरू देखाउनुहोस् जसको कमिटर जानकारी निर्दिष्ट स्ट्रिङसँग मेल खान्छ।
--grep	केवल ती कमिटहरू देखाउनुहोस् जसको कमिट सन्देशमा निर्दिष्ट स्ट्रिङ समावेश छ।
-S	केवल ती कमिटहरू देखाउनुहोस् जसले उक्त स्ट्रिङको कोड थप्यो वा हटायो।

Option	Description	Detailed Explanation
`-<n>`	Show only the last `n` commits.	This option is used to limit the output of the `git log` to the last `n` commits. You can replace `<n>` with any integer to show that many commits. For example, `git log -5` will show only the last 5 commits in the history.
`--since, --after`	Limit the commits to those made after the specified date.	This option allows you to specify a date (or a relative time) to filter commits that were made after that date. For example: `git log --since="2024-01-01"` will show commits made after January 1, 2024. You can also use relative times like `"2 weeks ago"`. This is useful when you want to see recent changes.
`--until, --before`	Limit the commits to those made before the specified date.	Similar to `--since`, but this option limits the commits to those that were made **before** the specified date. For example: `git log --before="2024-01-01"` will show commits made before January 1, 2024. You can also use relative time formats, such as `"1 month ago"`.
`--author`	Only show commits in which the author entry matches the specified string.	This option filters the commits to only those where the author matches the string you provide. For example, `git log --author="John Doe"` will only show commits authored by John Doe. This can be useful to review the work of a specific contributor in the repository.
`--committer`	Only show commits in which the committer entry matches the specified string.	This option filters commits by the committer's name or email address. It works similarly to `--author`, but instead of filtering by the author, it filters by the committer, which could be different from the author in the case of patch submissions. For example: `git log --committer="Jane Smith"`.
`--grep`	Only show commits with a commit message containing the specified string.	This option allows you to search commit messages for a specific string. For example, `git log --grep="bug fix"` will show only commits whose messages contain the string "bug fix". This is useful when you're looking for commits related to a particular feature or issue.
`-S`	Only show commits adding or removing code matching the specified string.	This option searches for changes that add or remove a specific string or piece of code. It is often called the "pickaxe" option. For example, `git log -S "function_name"` will show commits where the number of occurrences of the string "function_name" was modified, which is useful when tracking code changes.

मर्ज कमिटहरूको प्रदर्शन रोक्नु

तपाईंको रिपोजिटोरीमा प्रयोग गरिएको वर्कफ्लो अनुसार, तपाईंको लज्ज इतिहासमा ठूलो प्रतिशत मर्ज कमिटहरू हुन सक्छ जुन सामान्यतया धेरै जानकारीमूलक हुँदैन। तपाईंको `log` इतिहासलाई मर्ज कमिटहरूले अव्यवस्थित हुनबाट रोक्नको लागि, तपाईंले केवल `--no-merges` विकल्प थप्नुहोस्।

```
$ git log --no-merges
```

यसले मर्ज कमिटहरूलाई तपाईंको git log बाट हटाउँछ र केवल अन्य महत्त्वपूर्ण कमिटहरू देखाउँछ।

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