# ट्यागिंग (Tagging) - -

Git मा तपाईलाई एक निश्चित स्थानमा ट्याग गर्न सक्छ जुन इतिहासमा महत्त्वपूर्ण छ। प्रायः, यसलाई रिलिज बिन्दुहरू (जस्तै v1.0, v2.0) को रूपमा चिन्हित गर्न प्रयोग गरिन्छ। यस खण्डमा, तपाईले कसरी ट्यागहरू सूचीबद्ध गर्ने, सिर्जना गर्ने र मेट्ने, र विभिन्न प्रकारका ट्यागहरू के हुन् भनेर जान्नुहुनेछ।

---

## तपाईका ट्यागहरू सूचीबद्ध गर्नुहोस्

Git मा विद्यमान ट्यागहरू सूचीबद्ध गर्न सजिलो छ। केवल git tag कमाण्ड प्रयोग गर्नुहोस् (वैकल्पिक -l वा --list को साथ):

```
$ git tag
```

**नतिजा:**

```
v1.0
v2.0
```

यस कमाण्डले ट्यागहरूलाई वर्णानुक्रममा सूचीबद्ध गर्दछ; ती प्रदर्शित हुने क्रममा कुनै वास्तविक महत्त्व छैन।

---

## विशिष्ट ट्यागहरू खोज्नुहोस्

यदि तपाईसँग धेरै ट्यागहरू छन् भने, तपाई कुनै विशेष ढाँचासँग मेल खाने ट्यागहरू खोज्न सक्नुहुन्छ। उदाहरणका लागि, यदि तपाई v1.8.5 शृङ्खलामा मात्र चासो राख्नुहुन्छ भने, तपाई निम्न कमाण्ड चलाउन सक्नुहुन्छ:

**कमाण्ड:**

```
$ git tag -l "v1.8.5*"
```

यसले "v1.8.5" बाट सुरु हुने सबै ट्यागहरू देखाउनेछ, जसले तपाईलाई केवल प्रासंगिक संस्करणहरू हेर्न मद्दत पुर्याउँछ।

---

## ट्यागिङको सिंहावलोकन:

1. **ट्याग सूचीबद्ध गर्नुहोस्:**
   git tag वा git tag -l
2. **विशिष्ट ट्यागहरू खोज्नुहोस्:**
   git tag -l "pattern"

**ट्याग सूची वाइल्डकार्डको लागि -l वा --list अप्सन आवश्यक छ**
**यदि तपाई केवल सबै ट्यागहरूको सूची चाहनुहुन्छ भने, git tag कमाण्ड चलाउँदा यो स्वतः सूची प्रदान गर्दछ र -l वा --list को प्रयोग वैकल्पिक हुन्छ। तर, यदि तपाई ट्याग नामसँग मेल खाने वाइल्डकार्ड ढाँचाको आपूर्ति गर्दै हुनुहुन्छ भने, -l वा --list को प्रयोग अनिवार्य हुन्छ।**

# ट्यागहरू सिर्जना गर्नु (Creating tags) - -

Git ले दुई प्रकारका ट्यागहरू समर्थन गर्दछ: **lightweight** र **annotated**।

- **Lightweight tag**: यो एक शाखा जस्तै हुन्छ जुन परिवर्तन हुँदैन – यो एक विशिष्ट commit लाई मात्र सन्देश दिन्छ।
- **Annotated tag**: यी Git डाटाबेसमा पूर्ण वस्तुहरूको रूपमा सुरक्षित गरिन्छ। यी checksum गरिएको हुन्छन्, tag गर्नेको नाम, इमेल, र मिति समावेश गर्दछ, tagging सन्देश हुन्छ, र GNU Privacy Guard (GPG) सँग हस्ताक्षर र प्रमाणित गर्न सकिन्छ। सामान्यतया, annotated tags सिर्जना गर्नु सिफारिस गरिन्छ ताकि तपाईंलाई सबै जानकारी प्राप्त गर्न सकिन्छ। यद्यपि, यदि तपाईंलाई अस्थायी tag चाहिन्छ वा कुनै कारणले अन्य जानकारी राख्न चाहनुहुन्न भने, lightweight tags पनि उपलब्ध छन्।

## Annotated Tags

Git मा annotated tag सिर्जना गर्नु सरल छ। सबैभन्दा सजिलो तरिका भनेको `-a` प्रयोग गर्नु हो जब तपाईं tag आदेश चलाउनुहुन्छ:

```
$ git tag -a v1.4 -m "my version 1.4"
$ git tag
v0.1
v1.3
v1.4
```

यहाँ `-m` ले tagging सन्देश निर्दिष्ट गर्दछ, जुन tag सँग सुरक्षित गरिन्छ। यदि तपाईं annotated tag को लागि सन्देश निर्दिष्ट गर्नुहुन्न भने, Git तपाईंको सम्पादक खोल्दछ ताकि तपाईं त्यहाँ सन्देश लेख्न सक्नुहुन्छ।

तपाईं tag सँगै commit देख्नको लागि `git show` आदेश प्रयोग गर्न सक्नुहुन्छ:

```
$ git show v1.4
tag v1.4
Tagger: Ben Straub <ben@straub.cc>
Date: Sat May 3 20:19:12 2014 -0700
my version 1.4
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700
  Change version number
```

यसले tag गर्नेको जानकारी, commit tag गरिएको मिति, र annotation सन्देश देखाउँछ, र त्यसपछि commit को जानकारी देखाउँछ।

# Lightweight Tags

Git मा commit हरूलाई tag गर्ने अर्को तरिका lightweight tag हो। यो मूलतः commit checksum हो जुन एउटा फाइलमा सुरक्षित गरिएको हुन्छ – कुनै अन्य जानकारी राखिँदैन। lightweight tag सिर्जना गर्नको लागि, तपाईंले -a, -s, वा -m कुनै विकल्प पनि दिनुहुन्न, केवल tag को नाम मात्र दिनुहुन्छ:

```
$ git tag v1.4-lw
$ git tag
v0.1
v1.3
v1.4
v1.4-lw
v1.5
```

यसपटक, यदि तपाईं git show आदेश tag मा चलाउनुहुन्छ भने, तपाईंले अतिरिक्त tag जानकारी देख्नुहुन्न। आदेशले केवल commit देखाउँछ:

```
$ git show v1.4-lw
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date: Mon Mar 17 21:52:11 2008 -0700
  Change version number
```

---

**Tagging Later**

तपाईंले पछि गएर पनि commits लाई tag गर्न सक्नुहुन्छ। तपाईंको commit इतिहास यसरी देखिन सक्छ:

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 Create write support
0d52aaab4479697da7686c15f77a3d64d9165190 One more thing
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc Add commit function
4682c3261057305bdd616e23b64b0857d832627b Add todo file
166ae0c4d3f420721acbb115cc33848dfcc2121a Create write support
9fceb02d0ae598e95dc970b74767f19372d61af8 Update rakefile
964f16d36dfccde844893cac5b347e7b3d44abbc Commit the todo
8a5cbc430f1a9c3d00faaeffd07798508422908a Update readme
```

अब, तपाईंले v1.2 मा tag गर्न बिर्सनु भएको छ, जुन “Update rakefile” commit मा थियो। तपाईं यसलाई पछि पनि tag गर्न सक्नुहुन्छ। त्यसका लागि, तपाईंले commit checksum (वा यसको केही अंश) आदेशको अन्त्यमा दिइराख्नु पर्छ:

```
$ git tag -a v1.2 9fceb02
```

अब तपाईं देख्न सक्नुहुन्छ कि तपाईंले commit मा tag गर्नुभएको छ:

```
$ git tag
v0.1
v1.2
v1.3
```

```
v1.4
v1.4-lw
v1.5
```

र git show चलाउँदा tag को जानकारी यसरी देखिन्छ:

```
$ git show v1.2
tag v1.2
Tagger: Scott Chacon <schacon@gee-mail.com>
Date: Mon Feb 9 15:32:16 2009 -0800
version 1.2
commit 9fceb02d0ae598e95dc970b74767f19372d61af8
Author: Magnus Chacon <mchacon@gee-mail.com>
Date: Sun Apr 27 20:43:35 2008 -0700
  Update rakefile
```

यसरी तपाईंले पछिका commits मा पनि tags थप्न सक्नुहुन्छ।

---

# Sharing Tags ---

डिफल्ट रूपमा, git push आदेशले tags लाई remote server मा ट्रान्सफर गर्दैन। तपाईंले tags सिर्जना गरेपछि तिनीहरूलाई विशेष रूपमा साझा गर्नको लागि git push origin <tagname> चलाउनु पर्नेछ।

```
$ git push origin v1.5
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (14/14), 2.05 KiB | 0 bytes/s, done.
Total 14 (delta 3), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag] v1.5 -> v1.5
```

यदि तपाईं धेरै tags एक साथ पठाउन चाहनुहुन्छ भने, तपाईं git push आदेशसँग --tags विकल्प प्रयोग गर्न सक्नुहुन्छ। यसले सबै tags लाई remote server मा ट्रान्सफर गर्दछ जुन पहिले नै त्यहाँ छैनन्।

```
$ git push origin --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 160 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new tag] v1.4 -> v1.4
 * [new tag] v1.4-lw -> v1.4-lw
```

अब, जब अरू कोही तपाईंको repository लाई clone वा pull गर्छ, उनीहरूले तपाईंका सबै tags पनि प्राप्त गर्नेछन्।

**git push ले दुबै प्रकारका tags लाई पुश गर्दछ।**

**git push <remote> --tags ले lightweight र annotated दुबै प्रकारका tags लाई पुश गर्दछ। अहिले lightweight tags मात्र पुश गर्नको लागि कुनै विशेष विकल्प छैन, तर यदि तपाईं git push <remote> --follow-tags प्रयोग गर्नुहुन्छ भने, केवल annotated tags मात्र remote मा पुश गरिनेछन्।**

# Tags मेट्नको लागि: - -

तपाईं आफ्नो स्थानीय repository मा tag मेट्न git tag -d <tagname> प्रयोग गर्न सक्नुहुन्छ। उदाहरणको लागि, माथिको lightweight tag मेट्न हामी यसरी गर्न सक्छौं:

```
$ git tag -d v1.4-lw
Deleted tag 'v1.4-lw' (was e7d5add)
```

ध्यान दिनुहोस् कि यसले remote servers मा tag मेट्दैन। remote server बाट tag मेट्नका लागि दुई सामान्य तरिकाहरू छन्।

पहिलो तरिका git push <remote> :refs/tags/<tagname> हो:

```
$ git push origin :refs/tags/v1.4-lw
To /git@github.com:schacon/simplegit.git
 - [deleted] v1.4-lw
```

यहाँ, null value (कसैले समावेश नगरेको) लाई colon भन्दा अघि remote tag name मा पठाइएको छ, जसले tag लाई मेटिदिन्छ।

दोस्रो (र अधिक intuitive) तरिका भनेको git push origin --delete <tagname> हो:

```
$ git push origin --delete v1.4-lw
```

---

## Tags चेकआउट गर्ने:( Checking out Tags) ---

यदि तपाईं कुनै tag मा भएका फाइलको संस्करणहरू हेर्न चाहनुहुन्छ भने, तपाईं त्यस tag को git checkout गर्न सक्नुहुन्छ, यद्यपि यसले तपाईंको repository लाई "detached HEAD" स्थितिमा पुर्याउँछ, जसका केहि नकरात्मक प्रभावहरू हुन्छन्। उदाहरणका लागि:

```
$ git checkout v2.0.0
Note: switching to 'v2.0.0'.
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.
```

जब तपाईं detached HEAD स्थितिमा हुनुहुन्छ, तपाईंले परिवर्तनहरू गर्न र commit गर्न सक्छौं, र तपाईंले गरिएका commit हरूलाई discard गर्न पनि सक्नुहुन्छ, जसले कुनै पनि branches मा असर गर्दैन। यदि तपाईं नयाँ branch सिर्जना गर्न चाहनुहुन्छ भने, तपाईं -c का साथ git switch कमाण्ड प्रयोग गरेर यो गर्न सक्नुहुन्छ। उदाहरण:

```
git switch -c <new-branch-name>
```

या यस कार्यलाई undo गर्नको लागि:

```
git switch -
```

यदि तपाईं detached HEAD स्थितिमा रहँदा परिवर्तनहरू गर्न र commit गर्नुभयो भने, तपाईंको नयाँ commit पुरानो branch सँग जडान नगरी केवल tag को सन्दर्भमा रहनेछ, र नयाँ commit त्यस tag मा पुग्न सकिँदैन, यदि तपाईंले commit hash थाहा पाउनु भएको छैन भने। यसैले, यदि तपाईंले कुनै पुरानो संस्करणमा bug समाधान गर्दै हुनुहुन्छ भने, तपाईं सामान्यतया नयाँ branch बनाउने चाहनुहुन्छ:

```
$ git checkout -b version2 v2.0.0
Switched to a new branch 'version2'
```

अब तपाईंले commit गरेपछि, तपाईंको version2 branch पुरानो v2.0.0 tag भन्दा थोरै फरक हुनेछ, किनकि यसले तपाईंका नयाँ परिवर्तनहरू समेट्नेछ। त्यसैले यस अवस्थामा ध्यान दिनुहोस्।

---