Git Server Setup and Remote Repositories in Nepali

**Git Remote Repository को आवश्यकता**

Git मा सहकार्य (collaboration) गर्नको लागि **remote repository** चाहिन्छ।

* तपाईं व्यक्तिगत repositories मा परिवर्तनहरू push र pull गर्न सक्नुहुन्छ, तर यो राम्रो अभ्यास होइन।
* किनभने योले सहकार्यकर्ताहरूको काममा भ्रम उत्पन्न गराउन सक्छ।
* **Intermediate Repository** (जुन तपाईं र तपाईंका सहकार्यकर्ताले access गर्न सक्नुहुन्छ) प्रयोग गर्नु उचित हुन्छ।
* यसले तपाईंको सहकार्यकर्ताहरूलाई तपाईंको कम्प्युटर अफलाइन भए पनि repository पहुँच गर्न सक्षम बनाउँछ।

**Git Server चलाउनका चरणहरू**

Git server चलाउनका लागि यी चरणहरू अपनाउनुहोस्:

1. **प्रोटोकल चयन गर्नुहोस्**  
   Git server को लागि कुन प्रोटोकल प्रयोग गर्ने भन्ने निर्णय गर्नुहोस्।
   * **प्रोटोकलहरू**: HTTP(S), SSH, Git, आदि।
   * हरेक प्रोटोकलका आफ्ना फाइदाहरू र कमजोरीहरू छन्।
   * प्रोटोकलहरूको बारेमा विस्तृत जानकारी अध्यायको पहिलो भागमा दिइएको छ।
2. **टिपिकल सेटअप र Configuration**
   * प्रोटोकलहरूको आधारमा server को सेटअप र configuration गर्न सिक्नुहोस्।
   * उदाहरण: SSH आधारित Git server, HTTP server को Git integration आदि।
3. **Server Host गराउने विकल्पहरू**
   * यदि तपाईं आफ्नै server सेटअप गर्न चाहनुहुन्न भने, Git hosting providers (जस्तै GitHub, GitLab, Bitbucket) को प्रयोग गर्न सक्नुहुन्छ।
   * यसले तपाईंलाई server maintenance को झन्झटबाट बचाउँछ।

**Bare Repository के हो?**

Remote repository प्राय: **bare repository** हुन्छ।

* Bare repository भनेको Git repository हो, जसमा **working directory** हुँदैन।
* यस repository मा **snapshot (code checkout)** छैन।
* Bare repository केवल Git data सम्हाल्नका लागि प्रयोग गरिन्छ।
* **साधारण शब्दमा**:
  + यो तपाईंको प्रोजेक्टको **.git directory** को सामग्री मात्र हुन्छ, अरु केही हुँदैन।

**Hosted Remote Repository को फाइदा**

* Hosted repositories ले **सहज पहुँच** दिन्छन्।
* तपाईंलाई server को management गर्नुपर्दैन।
* उदाहरण:
  + **GitHub**: Public र Private repositories को लागि लोकप्रिय।
  + **GitLab**: Free private repositories र CI/CD integration को लागि राम्रो।
  + **Bitbucket**: Small teams को लागि उपयुक्त।

**समाप्ति सुझाव**

* यदि तपाईंलाई Git server सेटअप गर्न समय र झन्झटको समस्या छ भने, hosting providers को विकल्प चयन गर्नुहोस्।
* Hosting providers मा खाता बनाउनुहोस्, र आफ्नो कोड सुरक्षित रूपमा व्यवस्थापन गर्नुहोस्।
* अगाडिको अध्यायमा **distributed source control environment** मा सहकार्य गर्ने बिधिहरूको बारेमा चर्चा गरिनेछ।

**Git Protocols Overview in Nepali**

Git ले डेटा ट्रान्सफर गर्न चार वटा फरक प्रोटोकल प्रयोग गर्न सक्छ:

1. **Local Protocol**
2. **HTTP**
3. **Secure Shell (SSH)**
4. **Git Protocol**

**1. Local Protocol (स्थानिय प्रोटोकल)**

* **के हो?**  
  Local Protocol मा remote repository त्यहि host मा अर्को directory मा हुन्छ।
  + यो प्रोटोकल shared filesystem (जस्तै: NFS mount) भएका टिमहरूका लागि उपयोगी छ।
  + सबैले एउटै computer मा login गर्ने परिप्रेक्ष्यमा पनि प्रयोग गर्न सकिन्छ।  
    तर, यो आदर्श छैन, किनभने एउटै computer मा सबै instance राख्दा data को catastrophic loss हुन सक्छ।
* **कसरी काम गर्छ?**  
  यदि तपाईंको टीमले shared mounted filesystem प्रयोग गरिरहेको छ भने, तपाईं local file-based repository मा clone, push र pull गर्न सक्नुहुन्छ।
  + **Repository Clone गर्ने उदाहरण:**

$ git clone /srv/git/project.git

* + **File Protocol प्रयोग गरेर Clone गर्ने:**

$ git clone file:///srv/git/project.git

* **File Protocol र Path URL बीचको फरक:**
  + **Path URL** प्रयोग गर्दा Git ले hardlinks प्रयोग गर्छ वा फाइलहरू प्रत्यक्ष रूपमा copy गर्छ।
    - यो बढी छिटो हुन्छ।
  + **File Protocol (file://)** प्रयोग गर्दा Git ले data ट्रान्सफर गर्न नेटवर्क प्रक्रियाहरू सुरु गर्छ।
    - यो तुलनात्मक रूपमा कम प्रभावकारी हुन्छ।
    - तर, यो अन्य VCS बाट import गरेको repository को unnecessary references वा objects हटाउन उपयोगी छ।
* **सारांश:**
  + सामान्यतः **path URL** को प्रयोग गर्नु राम्रो हुन्छ, किनभने यो छिटो छ।
  + **file://** prefix तब प्रयोग गर्नुहोस्, जब तपाईंलाई repository को सफा प्रतिलिपि (clean copy) चाहिन्छ।

**Local Protocol कहिले प्रयोग गर्ने?**

* Shared filesystem (जस्तै: NFS) प्रयोग गर्ने condition मा।
* Single host मा काम गर्ने टिमहरूका लागि।
* Network-based प्रोटोकलहरू आवश्यक नभएको अवस्थामा।

Git मा कुनै पनि **स्थानिय रिपोजिटरी (Local Repository)** लाई एउटा प्रोजेक्टमा रिमोटको रूपमा थप्नको लागि, तपाईं निम्न कमाण्ड चलाउन सक्नुहुन्छ:

$ git remote add local\_proj /srv/git/project.git

**विवरण:**

* **git remote add**  
  यो कमाण्डले Git मा नयाँ रिमोट रिपोजिटरी थप्छ।
* **local\_proj**  
  यो तपाईंले दिएको रिमोट रिपोजिटरीको नाम हो। तपाईं यसलाई कुनै पनि नाम दिन सक्नुहुन्छ।
* **/srv/git/project.git**  
  यो स्थानिय रिपोजिटरीको path हो।

**अब के गर्न सकिन्छ?**

1. **Push गर्नुहोस्:**  
   तपाईं आफ्नो परिवर्तनहरू (changes) यो नयाँ रिमोटमा push गर्न सक्नुहुन्छ:

$ git push local\_proj branch\_name

1. **Pull गर्नुहोस्:**  
   तपाईं यो रिमोटबाट आफ्नो परिवर्तनहरू pull गर्न सक्नुहुन्छ:

$ git pull local\_proj branch\_name

**यसरी काम किन गर्ने?**

* यो प्रक्रिया तपाईंलाई स्थानिय रिपोजिटरीसँग काम गर्न दिन्छ जसरी तपाईं कुनै नेटवर्क-आधारित रिमोटसँग काम गर्नुहुन्छ।
* तपाईंले स्थानिय रिपोजिटरीलाई Git नेटवर्क जस्तो गरी प्रयोग गर्न सक्नुहुन्छ।

फाइल-आधारित रिपोजिटरीहरू सेटअप गर्न निकै सरल र सजिलो हुन्छन्, विशेष गरी यदि तपाईंको टोलीले पहिले नै साझा फाइल प्रणालीमा पहुँच राखेको छ भने। यसको सबैभन्दा ठूलो फाइदा यो हो कि तपाईंलाई कुनै विशेष सर्भर सेटअप आवश्यक पर्दैन। तपाईंले केवल रिपोजिटरीलाई एक साझा फोल्डरमा राख्नु पर्छ, जुन टोलीका सबै सदस्यले पहुँच गर्न सकून्।

यो प्रकारको रिपोजिटरीले पहिले नै रहेको फाइलको अनुमतिलाई (permissions) प्रयोग गर्छ। त्यसैले, पहुँच नियन्त्रण (जस्तै, पढ्ने वा लेख्ने अनुमति) सामान्य साझा फोल्डरमा जस्तो सजिलै व्यवस्थापन गर्न सकिन्छ।

यदि तपाईंसँग ठूलो टोली छैन र तपाईं छिटो काम साझेदारी गर्न चाहनुहुन्छ भने, फाइल-आधारित रिपोजिटरी विशेष उपयोगी हुन सक्छ। उदाहरणका लागि, तपाईंको सहकर्मीले कुनै फिचर जाँच गर्न आग्रह गरे भने, उनीहरूले आफ्नो परिवर्तनहरू रिमोट सर्भरमा पठाउनुभन्दा सिधै उनीहरूको रिपोजिटरीबाट तान्न (pull) सजिलो हुन्छ। यो प्रक्रिया निम्न प्रकारले गर्न सकिन्छ:

$ git pull /home/john/project

यसले साना टोलीहरूका लागि छिटो र प्रभावकारी सहयोग सुनिश्चित गर्दछ। तर ध्यान दिनुहोस्, यो विधि ठूला टोलीहरू वा लामो समयसम्म चल्ने प्रोजेक्टहरूका लागि आदर्श होइन, किनभने यो प्रकारको सेटअपमा केन्द्रीकृत सर्भरको जस्तो स्थायित्व र ट्र्याकिङ छैन।

यो विधिको कमजोरी भनेको साझा पहुँच (shared access) सेटअप गर्नु र विभिन्न स्थानबाट पहुँच गर्नु सामान्यतया नेटवर्क-आधारित पहुँचभन्दा गाह्रो हुनु हो। उदाहरणका लागि, यदि तपाईं घरमा हुँदा आफ्नो ल्यापटपबाट परिवर्तन पठाउन (push) चाहनुहुन्छ भने, तपाईंले पहिले रिमोट डिस्क माउन्ट गर्नुपर्ने हुन्छ। यो प्रक्रिया नेटवर्क-आधारित पहुँचको तुलनामा जटिल र ढिलो हुन सक्छ।

यसका अतिरिक्त, यदि तपाईं कुनै साझा माउन्ट गरिएको फाइल प्रणाली (जस्तै, NFS) प्रयोग गर्दै हुनुहुन्छ भने, यो विधि आवश्यक रूपमा सबैभन्दा छिटो छैन। स्थानीय रिपोजिटरी (local repository) केवल तब छिटो हुन्छ जब तपाईंले डेटा छिटो पहुँच गर्न सक्नुहुन्छ। उदाहरणका लागि, NFS मा रहेको रिपोजिटरी, त्यही सर्भरमा SSH प्रयोग गरेर गरिएको रिपोजिटरीभन्दा प्रायः ढिलो हुन्छ। किनभने SSH मा गिटले प्रत्येक प्रणालीका स्थानीय डिस्कहरूबाट काम गर्न सक्ने सुविधा दिन्छ।

अन्ततः, यो प्रोटोकलले रिपोजिटरीलाई दुर्घटनावश भएको क्षतिबाट जोगाउँदैन। प्रत्येक प्रयोगकर्तासँग "रिमोट" डायरेक्टोरीमा पूर्ण पहुँच हुन्छ। यो पहुँचले प्रयोगकर्ताहरूलाई गिटको आन्तरिक फाइलहरू परिवर्तन गर्न वा हटाउन अनुमति दिन्छ, जसले गर्दा रिपोजिटरी खराब (corrupt) हुन सक्छ। यस्तो जोखिमका कारण यो विधि ठूला वा दीर्घकालीन प्रोजेक्टहरूको लागि उपयुक्त छैन।

Git HTTP प्रोटोकल

Git HTTP प्रोटोकलको माध्यमबाट संवाद गर्ने दुई प्रमुख तरिका छन्। Git 1.6.6 भन्दा पहिले, यो प्रोटोकल केवल एक तरिकामा काम गर्थ्यो, जुन सामान्यतया केवल पढ्न योग्य (read-only) मात्र थियो। यस संस्करणमा Git ले केवल डेटा प्राप्त गर्ने काम गर्थ्यो, यसमा डेटा पठाउनको लागि कुनै सुविधा थिएन।

तर, Git 1.6.6 मा नयाँ र स्मार्ट प्रोटोकलको परिचय गरियो जसले Git लाई डेटा ट्रान्सफर गर्नको लागि एक स्मार्ट तरिकाले मोलतोल गर्न सक्षम बनायो। यो प्रक्रिया SSH प्रोटोकलसँग मिल्दोजुल्दो छ। यसले Git लाई अधिक दक्षतापूर्वक र लचिलो तरीकामा संवाद गर्न अनुमति दिन्छ।

पछिल्ला केही वर्षहरूमा यो नयाँ HTTP प्रोटोकल अत्यधिक लोकप्रिय भएको छ, र यो प्रयोगकर्ताका लागि सरल र स्मार्ट बनाएको छ। नयाँ प्रोटोकललाई "Smart HTTP प्रोटोकल" भनिन्छ भने पुरानो विधिलाई "Dumb HTTP" भनिन्छ। Smart HTTP प्रोटोकलले डेटा ट्रान्सफरको प्रक्रियालाई अधिक प्रभावकारी र लचिलो बनाउँछ, जसले Git प्रयोगकर्ताहरूको अनुभवलाई धेरै सुधार गर्दछ।

## Smart HTTP प्रोटोकल

Smart HTTP प्रोटोकल, SSH वा Git प्रोटोकल जस्तै काम गर्छ, तर यो सामान्य HTTPS पोर्टहरूमा चल्छ र यसले विभिन्न HTTP प्रमाणिकरण विधिहरू प्रयोग गर्न सक्छ। यसकारण, यो प्रायः SSH जस्तो जटिल सेटअपको तुलनामा प्रयोगकर्ताको लागि सजिलो हुन्छ, किनकि यहाँ तपाईंलाई SSH कुञ्जी सेटअप गर्नुको सट्टा यूजरनेम र पासवर्ड प्रमाणिकरण जस्ता सामान्य विधिहरू प्रयोग गर्न मिल्छ।

आजकल, यो प्रोटोकल Git प्रयोग गर्ने सबैभन्दा लोकप्रिय विधि बनेको छ, किनकि यसले git:// प्रोटोकल जस्तै अज्ञात पहुँचको लागि सेटअप गर्न सक्छ, र SSH प्रोटोकल जस्तै प्रमाणिकरण र इन्क्रिप्शनको साथ डेटा पठाउन सक्छ। यसको फाइदा यो हो कि तपाईंले यी सबै कार्यहरूको लागि अलग-अलग URL सेटअप गर्नुपर्दैन; तपाईं एकै URL प्रयोग गर्न सक्नुहुन्छ। यदि तपाईंले पुश गर्ने प्रयास गर्नुभयो र रिपोजिटोरीले प्रमाणिकरणको आवश्यकता छ भने (जुन सामान्यतया हुनु पर्छ), त्यसपछि सर्भरले तपाईंलाई यूजरनेम र पासवर्डको लागि अनुरोध गर्न सक्छ। यसले पढ्नको लागि पहुँच पनि यस्तै गर्छ।

असली उदाहरणको रूपमा, GitHub जस्ता सेवाहरूमा, तपाईंले अनलाइन रिपोजिटोरी हेर्नको लागि जुन URL प्रयोग गर्नुहुन्छ (जस्तै, [**https://github.com/schacon/simplegit**](https://github.com/schacon/simplegit)), त्यही URL तपाईंले क्लोन गर्न र यदि तपाईंको पहुँच छ भने, पुश गर्न पनि प्रयोग गर्न सक्नुहुन्छ।

## Dumb HTTP प्रोटोकल

यदि सर्भरले Git HTTP स्मार्ट सेवा प्रदान नगरेको खण्डमा, Git क्लाइन्टले सरल Dumb HTTP प्रोटोकलमा फर्किने प्रयास गर्नेछ। Dumb प्रोटोकलले Gitको बियर रिपोजिटोरीलाई सामान्य फाइलको रूपमा वेब सर्भरबाट सेवा गर्न अपेक्षा गर्दछ। Dumb HTTP को सौन्दर्य यसको सेटअपको साधारणतामा छ। यो सेटअप गर्नका लागि तपाईंलाई के गर्नुपर्नेछ भने, बियर Git रिपोजिटोरीलाई तपाईंको HTTP डोक्युमेन्ट रुटमा राख्नु हो र एक विशेष पोस्ट-अपडेट हुक सेटअप गर्नु हो। यसपछि, जोसुकै व्यक्तिले वेब सर्भरमा राखिएको रिपोजिटोरीमा पहुँच प्राप्त गर्न सक्छ, उसले तपाईंको रिपोजिटोरी क्लोन गर्न सक्छ।

HTTP मार्फत तपाईंको रिपोजिटोरीमा पठाउनको लागि पढ्ने अनुमति दिनको लागि तपाईंले निम्न जस्ता आदेशहरू चलाउन सक्नुहुन्छ:

$ cd /var/www/htdocs/

$ git clone --bare /path/to/git\_project gitproject.git

$ cd gitproject.git

$ mv hooks/post-update.sample hooks/post-update

$ chmod a+x hooks/post-update

यो सबै हो। Gitसँगको डिफल्ट पोस्ट-अपडेट हुकले उचित आदेश (git update-server-info) चलाउँछ जसले HTTP मार्फत फेचिङ र क्लोनिङलाई ठीकसँग काम गर्न अनुमति दिन्छ। यो आदेश त्यसपछि चलाइन्छ जब तपाईं यस रिपोजिटोरीमा पुश गर्नुहुन्छ (शायद SSH मार्फत); त्यसपछि अन्य व्यक्तिहरू यसलाई यस प्रकार क्लोन गर्न सक्छन्:

$ git clone https://example.com/gitproject.git

यस विशेष केसमा, हामी /var/www/htdocs पथ प्रयोग गर्दैछौं जुन Apache सेटअपहरूको लागि सामान्य छ, तर तपाईं कुनै पनि स्थिर वेब सर्भर प्रयोग गर्न सक्नुहुन्छ — केवल बियर रिपोजिटोरीलाई यसको पथमा राख्नुहोस्। Git डेटा सामान्य स्थिर फाइलहरूको रूपमा सेवा गरिन्छ (Git इन्टरनल्स अध्यायमा यसको बारेमा विस्तृत जानकारी प्राप्त गर्न सक्नुहुन्छ)।

सामान्यतया, तपाईंले वा त एक पढ्न/लेखन स्मार्ट HTTP सर्भर चलाउने निर्णय गर्नुहुन्छ वा केवल Dumb तरिकामा फाइलहरूलाई पढ्नको लागि पहुँच योग्य बनाउनुहुन्छ। यो दुबै सेवाहरूको मिश्रण चलाउनु दुर्लभ हुन्छ।

**Smart HTTP प्रोटोकलका फाइदाहरू**

Smart HTTP प्रोटोकलको फाइदाहरूलाई यहाँ विस्तृत रूपमा व्याख्या गरिन्छ।

Smart HTTP को प्रमुख फाइदा यसको सरलता हो। सबै प्रकारका एक्सेसका लागि एउटै URL प्रयोग गर्न सकिने र जब प्रमाणीकरण आवश्यक हुन्छ मात्र सर्भरले प्रयोगकर्तालाई संकेत गर्ने प्रणालीले अन्तिम प्रयोगकर्ताका लागि यो सजिलो बनाउँछ। SSH भन्दा ठूलो फाइदा यो हो कि यसमा प्रयोगकर्ताले स्थानीय रूपमा SSH कीहरू बनाउनु पर्दैन र आफ्नो सार्वजनिक की सर्भरमा अपलोड गर्नु पर्ने झन्झट हुँदैन। यो कम प्राविधिक प्रयोगकर्ताहरूका लागि अथवा त्यस्ता प्रणालीहरूमा जहाँ SSH कम प्रयोग गरिन्छ, एक प्रमुख फाइदा हो।

यो प्रोटोकल छिटो र प्रभावकारी पनि छ, SSH प्रोटोकलसँग मिल्दोजुल्दो गतिमा काम गर्न सक्छ। HTTPS मार्फत तपाईंका रिपोजिटोरीहरूलाई केवल पढ्न (read-only) सेवा गर्न सकिन्छ, जसले डेटा ट्रान्सफरलाई इन्क्रिप्ट गर्न सम्भव बनाउँछ। थप रूपमा, तपाईं ग्राहकहरूलाई विशेष रूपमा हस्ताक्षरित SSL प्रमाणपत्रहरू प्रयोग गर्न बाध्य बनाउन सक्नुहुन्छ।

अर्को महत्वपूर्ण फाइदा भनेको HTTP र HTTPS यति सामान्य रूपमा प्रयोग गरिने प्रोटोकलहरू हुन् कि कर्पोरेट फायरवालहरू अक्सर तिनका पोर्टहरूमा ट्राफिक अनुमति दिन सेट गरिएका हुन्छन्। यसले कार्यस्थलहरू वा फायरवाल प्रतिबन्धहरू भएका ठाउँहरूमा पनि सहज रूपमा काम गर्न मद्दत गर्दछ।

**Smart HTTP प्रोटोकलका कमजोरीहरू**

Smart HTTP प्रोटोकलको उपयोग गर्दा केही कमजोरीहरू पनि देखिन्छन्।  
HTTPS मार्फत Git सेट अप गर्नु केही सर्भरहरूमा SSH भन्दा अलि जटिल हुन सक्छ। यसले गर्दा, सुरुवाती सेटअपमा समय र मेहनत अलि बढी लाग्न सक्छ।

यदि तपाईं HTTP मार्फत प्रमाणीकरण गरिएको पुश (authenticated pushing) गर्दै हुनुहुन्छ भने, क्रेडेन्शियल्स (जस्तै, प्रयोगकर्ता नाम र पासवर्ड) दिनु SSH को keys प्रयोग गरेभन्दा अलि बढी जटिल हुन सक्छ। यद्यपि, यस समस्या समाधान गर्नका लागि धेरै क्रेडेन्शियल क्यासिङ उपकरणहरू उपलब्ध छन्। उदाहरणका लागि, macOS मा **Keychain Access** र Windows मा **Credential Manager** जस्ता उपकरणहरू प्रयोग गर्न सकिन्छ, जसले यस प्रक्रिया सजिलो बनाउँछ।

यदि तपाईं यस प्रक्रियालाई थप सुरक्षित र छिटो बनाउन चाहनुहुन्छ भने, Credential Storage सेटअप गर्ने बारे सिक्नुपर्छ। यसले तपाईंको HTTP पासवर्ड क्यासिङलाई सुरक्षित बनाउनेछ।  
यद्यपि यी केही जटिलताहरू बाहेक, अन्य प्रोटोकलहरूसँग तुलना गर्दा Smart HTTP का कमजोरीहरू धेरै कम देखिन्छन्।

SSH प्रोटोकल - - -

Git मा, विशेष गरी जब तपाईं आफैंले होस्ट गरेको सर्भरको प्रयोग गरिरहनुभएको छ, SSH (Secure Shell) प्रोटोकल एउटा साधारण र प्रचलित ट्रान्सपोर्ट विधि हो। यसको मुख्य कारण यो हो कि अधिकांश स्थानहरूमा SSH मार्फत सर्भर पहुँचको व्यवस्था पहिल्यै गरिएको हुन्छ। यदि त्यो व्यवस्था छैन भने पनि, सेटअप गर्न निकै सजिलो हुन्छ।

SSH प्रोटोकलको अर्को फाइदा यो हो कि यो प्रमाणीकरण गरिएको (authenticated) नेटवर्क प्रोटोकल हो, जसले सर्भर र क्लाइन्टबीच सुरक्षित संचार सुनिश्चित गर्दछ। साथै, यसको व्यापक प्रयोगका कारण, सेटअप र प्रयोग दुवै धेरै सहज हुन्छ।

**Git Repository Clone गर्न SSH प्रयोग**

Git repository लाई SSH मार्फत clone गर्न तपाईंले ssh:// URL प्रयोग गर्न सक्नुहुन्छ:

$ git clone ssh://[user@]server/project.git

यसको अलावा, SSH प्रोटोकलका लागि छोटो र सुविधाजनक scp-जस्तो syntax पनि प्रयोग गर्न सकिन्छ:

$ git clone [user@]server:project.git

**Default Username**

यदि माथिका कुनै पनि विधिमा तपाईँले username तोकेनहुने हो भने, Git ले स्वतः तपाईं हालसालै login भएको प्रयोगकर्ताको नामलाई default username मान्छ।

SSH को यो विधि प्रायः Git को लागि सबैभन्दा लोकप्रिय र सुरक्षित विकल्प मानिन्छ, किनभने यो धेरैजसो सर्भरहरूमा पहिल्यै तयार अवस्थामा हुन्छ र सेटअप गर्न सजिलो छ।

**SSH प्रयोगका फाइदाहरू**

SSH प्रोटोकल प्रयोग गर्दा धेरै लाभहरू पाइन्छन्:

1. **सजिलो सेटअप:**  
   SSH सेटअप गर्न तुलनात्मक रूपमा सजिलो हुन्छ। धेरैजसो सर्भरहरूमा SSH daemon पहिल्यै सेटअप गरिएको हुन्छ, र नेटवर्क प्रशासकहरूलाई यसमा काम गर्ने राम्रो अनुभव हुन्छ। साथै, धेरैजसो अपरेटिङ सिस्टम वितरणहरूले SSH व्यवस्थापन गर्न उपकरणहरू प्रदान गर्छन्।
2. **सुरक्षित पहुँच:**  
   SSH मार्फत गरिएको सबै डेटा स्थानान्तरण एनक्रिप्टेड (encrypted) र प्रमाणीकरण गरिएको (authenticated) हुन्छ। यसले सुरक्षा सुनिश्चित गर्दछ, जसले गर्दा बाहिरी आक्रमणकर्ताले डेटा हेर्न वा चोर्न सक्दैन।
3. **दक्षता (Efficiency):**  
   HTTPS, Git, र Local प्रोटोकलहरूको जस्तै, SSH पनि अत्यन्त प्रभावकारी छ। डेटा पठाउनु अघि यसलाई यथासम्भव संकुचित (compact) बनाइन्छ, जसले गर्दा कम ब्यान्डविथमा पनि डेटा चाँडो पठाउन सकिन्छ।

यी सबै कारणहरूले गर्दा SSH Git repository व्यवस्थापनका लागि सुरक्षित, सजिलो र भरपर्दो विकल्प मानिन्छ।

**SSH प्रयोगका कमजोरीहरू**

SSH प्रोटोकल प्रयोग गर्दा केही सीमा वा कमजोरीहरू पनि छन्:

1. **अनामिक पहुँचको अभाव:**  
   SSH ले Git repository मा अनामिक (anonymous) पहुँचको समर्थन गर्दैन। यदि तपाईं SSH प्रयोग गर्दै हुनुहुन्छ भने, प्रयोगकर्ताहरूलाई तपाईंको सर्भरमा SSH पहुँच हुनु जरुरी हुन्छ, चाहे त्यो केवल read-only पहुँचका लागि किन नहोस्। यसले SSH लाई ती ओपन-सोर्स प्रोजेक्टहरूका लागि उपयुक्त बनाउँदैन जहाँ मानिसहरूले सरल रूपमा repository क्लोन गरी अध्ययन गर्न चाहन्छन्।
2. **खुला पहुँचको लागि असुविधा:**  
   यदि तपाईंले आफ्नो प्रोजेक्टहरूमा anonymous read-only पहुँच दिन चाहनुहुन्छ भने, SSH मात्र पर्याप्त हुँदैन। यस अवस्थामा, तपाईंलाई आफ्नो repository मा push गर्नका लागि SSH सेटअप गर्नुपर्नेछ, तर fetch (डाटा ल्याउने) गर्नका लागि अन्य प्रोटोकल प्रयोग गर्नुपर्ने हुन्छ।
3. **सीमित प्रयोगको सम्भावना:**  
   SSH प्रायः भित्री कर्पोरेट नेटवर्कमा मात्र उपयोगी हुन्छ, जहाँ प्रयोगकर्ताहरू पहिल्यै चिनिएका हुन्छन्। सार्वजनिक प्रयोगका लागि, विशेष गरी जहाँ सहज पहुँच आवश्यक हुन्छ, SSH एकमात्र विकल्प उपयुक्त नहुन सक्छ।

यी सीमाहरूका कारण SSH सामान्यतया सुरक्षित र कुशल भए पनि, विशिष्ट परिदृश्यहरूका लागि यसलाई अन्य प्रोटोकलहरूसँग मिलाएर प्रयोग गर्नुपर्ने आवश्यकता पर्न सक्छ।

Git प्रोटोकल - - -

Git प्रोटोकल Git को एउटा विशेष सेवा हो जसले Git रिपोजिटोरीहरूलाई सजिलो र छिटो सेवा दिन्छ। यो सेवा Git सँगै आउने daemon (पृष्ठभूमिमा चल्ने सेवा) को माध्यमबाट सञ्चालन हुन्छ।

यो प्रोटोकल एउटा निश्चित पोर्ट (9418) मा चल्छ र SSH प्रोटोकलसँग मिल्दोजुल्दो सेवा प्रदान गर्छ, तर यसमा कुनै पनि प्रमाणिकरण (authentication) वा क्रिप्टोग्राफी (encryption) हुँदैन। यदि तपाईंले कुनै रिपोजिटोरीलाई Git प्रोटोकलमार्फत सेवा दिन चाहनुहुन्छ भने, त्यो रिपोजिटोरीमा git-daemon-export-ok नामको फाइल राख्न आवश्यक पर्छ। यो फाइल राखिएन भने daemon ले रिपोजिटोरीलाई सेवा दिन सक्दैन।

यो प्रोटोकलमा कुनै सुरक्षा व्यवस्था छैन। जसले गर्दा, यदि रिपोजिटोरी सेवा उपलब्ध छ भने, जुनसुकैले त्यसलाई क्लोन गर्न सक्छ। तर, यो प्रोटोकल प्रायः लेखन अधिकार (write access) को लागि प्रयोग गरिँदैन। लेखन पहुँच उपलब्ध गराउन सकिए पनि, यसको कुनै प्रमाणिकरण नहुँदा कोही पनि इन्टरनेटमा उक्त प्रोजेक्टको URL पाएर त्यसलाई परिवर्तन गर्न सक्छ। यस्तो जोखिमको कारण, यो सुविधा बिरलै प्रयोग गरिन्छ।

Git प्रोटोकल मुख्यतया ती परियोजनाहरूका लागि उपयुक्त छ, जुन सार्वजनिक रुपमा पढ्न मात्रको लागि उपलब्ध गराउनुपर्छ। यो छिटो र सजिलो सेवा दिनका लागि डिजाइन गरिएको हो। तर, जहाँ सुरक्षा वा प्रयोगकर्ताको प्रमाणिकरण चाहिन्छ, त्यहाँ यो प्रोटोकल उपयुक्त हुँदैन।

यसकारण, Git प्रोटोकल ती अवस्थामा प्रयोग गरिन्छ जहाँ सार्वजनिक पहुँचको आवश्यकता छ र सुरक्षाको आवश्यकता छैन।

## **Git प्रोटोकलका फाइदा र कमजोरीहरू**

**फाइदाहरू**  
Git प्रोटोकललाई अक्सर सबैभन्दा छिटो नेटवर्क ट्रान्सफर प्रोटोकल मानिन्छ। यदि तपाईं कुनै सार्वजनिक परियोजनाको लागि धेरै ट्राफिक सेवा दिनुहुन्छ वा ठूलो परियोजनाको सेवा दिनुहुन्छ जहाँ प्रयोगकर्ताको प्रमाणिकरण आवश्यक छैन, यो प्रोटोकल निकै उपयोगी हुन्छ। यसले SSH प्रोटोकलसँग उस्तै डेटा ट्रान्सफर मेकानिज्म प्रयोग गर्छ तर क्रिप्टोग्राफी र प्रमाणिकरण जस्ता अतिरिक्त प्रक्रियाहरू बिना। त्यसैले यो धेरै छिटो छ।

**कमजोरीहरू**  
Git प्रोटोकलमा TLS (Transport Layer Security) वा अन्य क्रिप्टोग्राफीको अभाव छ, जसले गर्दा यो प्रोटोकल प्रयोग गर्दा सुरक्षा जोखिमहरू हुन्छन्।  
यदि तपाईंले git://example.com/project.git चलाउनुहुन्छ भने, कुनै आक्रमणकारी (जस्तै, तपाईंको राउटरमा नियन्त्रण भएको व्यक्तिले) तपाईंले क्लोन गरेको रिपोजिटोरीमा हानिकारक कोड राख्न सक्छ। यदि तपाईं त्यस कोडलाई चलाउनुहुन्छ वा कम्पाइल गर्नुहुन्छ भने, तपाईंले हानिकारक कोड पनि चलाउन सक्नुहुन्छ।

त्यसैले http:// को सट्टा https:// प्रयोग गर्न सिफारिस गरिन्छ, किनभने TLS प्रमाणपत्र बिना https:// लाई ह्याक गर्न गाह्रो हुन्छ।

यस प्रोटोकलमा कुनै प्रमाणिकरण छैन, जसको अर्थ, जसले पनि रिपोजिटोरी क्लोन गर्न सक्छ। यद्यपि, केही अवस्थामा यही चाहिएको पनि हुन सक्छ।

Git प्रोटोकल सेटअप गर्न गाह्रो पनि छ। यसलाई चलाउन अलग daemon चाहिन्छ, जसको लागि xinetd वा systemd जस्ता टुलहरूको कन्फिगरेसन आवश्यक पर्छ। यो प्रक्रियालाई सजिलो मानिँदैन। साथै, यसले पोर्ट 9418 को पहुँच चाहिन्छ, जुन सामान्यतया धेरै कर्पोरेट फायरवालहरूमा ब्लक गरिएको हुन्छ।

यसका कारण, ठूलो संस्थाका फायरवालहरू पछाडि Git प्रोटोकल प्रयोग गर्न समस्या हुन सक्छ।

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