प्रॉक्सी चेक का इम्प्लीमेंटेशन विंडोज में

□□□-□□□□ और □□□□□□□□□□ में □□□□□□□ पर प्रॉक्सी चेक लागू करने के लिए, प्रत्येक शेल में नेटवर्क संबंधी कमांडों को खोजने और प्रॉक्सी सेटिंग्स को दिखाने के लिए योजनाएं बनानी पड़ती हैं। नीचे □□□-□□□□ और □□□□□□□□□□ के लिए चरण और कोड दिए गए हैं। "टर्मिनल"का उल्लेख □□□□□□□ टर्मिनल में इन शेल्स को होस्ट करने के लिए किया गया है, इसलिए हम □□□-□□□□ और □□□□□□□□□□ के कार्यान्वयन पर ध्यान केंद्रित करेंगे।

□□□-□□□□ के लिए

□□□-□□□□ एक □□□□□□□ पर □□□□ एमुलेशन है, और हम एक DEBUG ट्रैप का उपयोग करके एक फंक्शन को प्रत्येक कमांड के कार्यान्वयन से पहले चलाने के लिए सेट अप कर सकते हैं। लक्ष्य यह है कि कमांड नेटवर्क संबंधी है और प्रॉक्सी सेटिंग्स सेट हैं, तो उन्हें दिखाएं।

चरण:

1. **नेटवर्क संबंधी कमांडों की सूची को परिभाषित करें।**
2. **प्रॉक्सी सेटिंग्स को दिखाने के लिए एक फंक्शन बनाएं।**
3. **कमांड और प्रॉक्सी सेटिंग्स को चेक करने के लिए एक फंक्शन बनाएं।**
4. `DEBUG` **ट्रैप को सेट करें ताकि चेक प्रत्येक कमांड से पहले चल सके।**
5. **प्रॉक्सी सेटिंग्स को मैनुअल रूप से दिखाने के लिए एक** `checkproxy` **फंक्शन परिभाषित करें।**
6. **सभी कॉन्फ़िगरेशन को** `.bashrc` **फ़ाइल में जोड़ें।**

कार्यान्वयन: निम्न कोड को आपकी `~/.bashrc` फ़ाइल में जोड़ें (यदि यह मौजूद नहीं है तो इसे बनाएं):

```
#
network_commands=(
    "gpa"
    "git"
    "ssh"
    "scp"
    "sftp"
    "rsync"
    "curl"
    "wget"
    "apt"
    "yum"
    "dnf"
    "npm"
    "yarn"
    "pip"
```

```
    "pip3"
    "gem"
    "cargo"
    "docker"
    "kubectl"
    "ping"
    "traceroute"
    "netstat"
    "ss"
    "ip"
    "ifconfig"
    "dig"
    "nslookup"
    "nmap"
    "telnet"
    "ftp"
    "nc"
    "tcpdump"
    "adb"
    "bundle"
    "brew"
    "cpanm"
    "bundle exec jekyll"
    "make"
    "python"
    "glcoud"
)

#
display_proxy() {
    echo -e "  **           :**"
    [ -n "$HTTP_PROXY" ] && echo "   - HTTP_PROXY: $HTTP_PROXY"
    [ -n "$http_proxy" ] && echo "   - http_proxy: $http_proxy"
    [ -n "$HTTPS_PROXY" ] && echo "   - HTTPS_PROXY: $HTTPS_PROXY"
    [ -n "$https_proxy" ] && echo "   - https_proxy: $https_proxy"
    [ -n "$ALL_PROXY" ] && echo "   - ALL_PROXY: $ALL_PROXY"
    [ -n "$all_proxy" ] && echo "   - all_proxy: $all_proxy"
    echo ""
}
```

```
#
proxy_check() {
    local cmd
    #
    cmd=$(echo "$BASH_COMMAND" | awk '{print $1}')

    for network_cmd in "${network_commands[@]}"; do
        if [[ "$cmd" == "$network_cmd" ]]; then
            #
            if [ -n "$HTTP_PROXY" ] || [ -n "$http_proxy" ] || \
               [ -n "$HTTPS_PROXY" ] || [ -n "$https_proxy" ] || \
               [ -n "$ALL_PROXY" ] || [ -n "$all_proxy" ]; then
                display_proxy
            fi
            break
        fi
    done
}

# `DEBUG`              `proxy_check`
trap 'proxy_check' DEBUG

#
checkproxy() {
    echo "HTTP_PROXY: $HTTP_PROXY"
    echo "HTTPS_PROXY: $HTTPS_PROXY"
    echo "Git HTTP Proxy:"
    git config --get http.proxy
    echo "Git HTTPS Proxy:"
    git config --get https.proxy
}
```

यह कैसे काम करता है:

- `network_commands` एक नेटवर्क संबंधी कमांडों की सूची है।
- `display_proxy` सभी प्रासंगिक प्रॉक्सी पर्यावरण चरों को दिखाता है यदि वे सेट हैं।
- `proxy_check` `BASH_COMMAND` (जो `DEBUG` ट्रैप में उपलब्ध है) का उपयोग करता है ताकि चलने वाली कमांड को प्राप्त कर सके, पहला शब्द निकाल सके और यह देख सके कि यह किसी नेटवर्क कमांड से मेल खाता है। यदि प्रॉक्सी चर सेट हैं, तो यह उन्हें दिखाता है।
- `trap 'proxy_check' DEBUG` पंक्ति सुनिश्चित करती है कि `proxy_check` प्रत्येक कमांड से पहले चलता है।
- `checkproxy` आपको प्रॉक्सी सेटिंग्स को मैनुअल रूप से देखने की अनुमति देता है, जिसमें □□□-वशिष्ट प्रॉक्सी कॉन्फ़िगरेशन शामिल हैं।

 .bashrc में यह जोड़ने के बाद, □□□-□□□□ को पुनः शुरू करें या source ~/.bashrc चलाएं ताकि परिवर्तन लागू हों।

उपयोग:

 जब आप एक नेटवर्क कमांड (जैसे git clone, curl) चलाते हैं, यदि प्रॉक्सी सेटिंग्स कॉन्फ़िगर किए गए हैं, तो वे कमांड के कार्यान्वयन से पहले दिखाए जाएंगे।
 checkproxy चलाएं ताकि प्रॉक्सी सेटिंग्स को मैनुअल रूप से देखें।

□□□□□□□□□□ **के लिए**

□□□□□□□□□□ में □□□□ की DEBUG ट्रैप का सीधा समकक्ष नहीं है, लेकिन हम PSReadLine मॉड्यूल के CommandValidationHandler का उपयोग करके समान कार्यान्वयन प्राप्त कर सकते हैं। यह हैंडलर प्रत्येक कमांड से पहले चलता है, जिससे हम नेटवर्क कमांडों और प्रॉक्सी सेटिंग्स को चेक कर सकते हैं।

चरण:

1. **नेटवर्क संबंधी कमांडों की सूची को परिभाषित करें।**
2. **प्रॉक्सी सेटिंग्स को दिखाने के लिए एक फंक्शन बनाएं।**
3. CommandValidationHandler **को सेट करें ताकि कमांड और प्रॉक्सी सेटिंग्स को चेक कर सके।**
4. **प्रॉक्सी सेटिंग्स को मैनुअल रूप से दिखाने के लिए एक** checkproxy **फंक्शन परिभाषित करें।**
5. **सभी कॉन्फ़िगरेशन को आपकी □□□□□□□□□□ प्रोफ़ाइल में जोड़ें।**

कार्यान्वयन: पहले, □□□□□□□□□□ में $PROFILE चलाएं ताकि आपकी □□□□□□□□□□ प्रोफ़ाइल फ़ाइल को खोज सकें। यदि यह मौजूद नहीं है, तो इसे बनाएं:

```
New-Item -Type File -Force $PROFILE
```

निम्न कोड को आपकी □□□□□□□□□□ प्रोफ़ाइल में जोड़ें (जैसे Microsoft.PowerShell_profile.ps1):

```
#
$networkCommands = @(
    "gpa",
    "git",
    "ssh",
    "scp",
    "sftp",
    "rsync",
    "curl",
```

```
    "wget",
    "apt",
    "yum",
    "dnf",
    "npm",
    "yarn",
    "pip",
    "pip3",
    "gem",
    "cargo",
    "docker",
    "kubectl",
    "ping",
    "traceroute",
    "netstat",
    "ss",
    "ip",
    "ifconfig",
    "dig",
    "nslookup",
    "nmap",
    "telnet",
    "ftp",
    "nc",
    "tcpdump",
    "adb",
    "bundle",
    "brew",
    "cpanm",
    "bundle exec jekyll",
    "make",
    "python",
    "glcoud"
)

#
function Display-Proxy {
    Write-Host "  **           :**"
    if ($env:HTTP_PROXY) { Write-Host "    - HTTP_PROXY: $env:HTTP_PROXY" }
    if ($env:http_proxy) { Write-Host "    - http_proxy: $env:http_proxy" }
```

```
    if ($env:HTTPS_PROXY) { Write-Host "   - HTTPS_PROXY: $env:HTTPS_PROXY" }
    if ($env:https_proxy) { Write-Host "   - https_proxy: $env:https_proxy" }
    if ($env:ALL_PROXY) { Write-Host "   - ALL_PROXY: $env:ALL_PROXY" }
    if ($env:all_proxy) { Write-Host "   - all_proxy: $env:all_proxy" }
    Write-Host ""
}

#               `CommandValidationHandler`
Set-PSReadLineOption -CommandValidationHandler {
    param($command)
    #
    $cmd = ($command -split ' ')[0]

    if ($networkCommands -contains $cmd) {
        #
        if ($env:HTTP_PROXY -or $env:http_proxy -or $env:HTTPS_PROXY -or $env:https_proxy -or $env:ALL_PROXY -
            Display-Proxy
        }
    }
    #                         $true
    return $true
}

#
function checkproxy {
    Write-Host "HTTP_PROXY: $env:HTTP_PROXY"
    Write-Host "HTTPS_PROXY: $env:HTTPS_PROXY"
    Write-Host "Git HTTP Proxy:"
    git config --get http.proxy
    Write-Host "Git HTTPS Proxy:"
    git config --get https.proxy
}
```

यह कैसे काम करता है:

- `$networkCommands` एक नेटवर्क संबंधी कमांडों की सूची है।
- `Display-Proxy` सभी प्रासंगिक प्रॉक्सी पर्यावरण चरों को दिखाता है यदि वे सेट हैं।
- `Set-PSReadLineOption -CommandValidationHandler` एक स्क्रिप्ट ब्लॉक परिभाषित करता है जो प्रत्येक कमांड से पहले चलता है:
  - यह कमांड का पहला शब्द निकालता है।
  - यह देखता है कि यह `$networkCommands` में है।

- यदि प्रॉक्सी चर सेट हैं, तो यह `Display-Proxy` को कॉल करता है।
- कमांड को चलाने की अनुमति देने के लिए `$true` लौटाता है।

- `checkproxy` आपको प्रॉक्सी सेटिंग्स को मैनुअल रूप से देखने की अनुमति देता है, जिसमें □□□-वशिष्ट प्रॉक्सी शामिल हैं।
- प्रोफ़ाइल में जोड़ने के बाद, □□□□□□□□□□ को पुनः शुरू करें या `. $PROFILE` चलाएं ताकि परिवर्तन लागू हों।

आवश्यकताएं:

- `PSReadLine` मॉड्यूल आवश्यक है, जो □□□□□□□□□□ 5.1 और बाद में डिफ़ॉल्ट रूप से शामिल है।
- अगर आप एक पुरानी संस्करण का उपयोग कर रहे हैं, तो □□□□□□□□□□ को अपग्रेड करना पड़ेगा या एक विकल्प ढूंढना पड़ेगा (यहाँ कवर नहीं किया गया है, क्योंकि अधिकांश प्रणालियाँ नए संस्करण का उपयोग करती हैं।

उपयोग:

- जब आप एक नेटवर्क कमांड (जैसे `git pull`, `curl`) चलाते हैं, यदि प्रॉक्सी सेटिंग्स कॉन्फ़िगर किए गए हैं, तो वे कमांड के कार्यान्वयन से पहले दिखाए जाएंगे।
- `checkproxy` चलाएं ताकि प्रॉक्सी सेटिंग्स को मैनुअल रूप से देखें।

टर्मिनल पर नोट्स

- अगर "टर्मिनल"□□□□□□□□ टर्मिनल का उल्लेख करता है, तो यह बस □□□-□□□□, □□□□□□□□□□ या कमांड प्रोम्प्ट (□□□.□□□) जैसे शेल्स को होस्ट करता है।
- ऊपर दिए गए कार्यान्वयन □□□□□□□□ टर्मिनल में □□□-□□□□ या □□□□□□□□□□ सेशन में काम करते हैं।
- कमांड प्रोम्प्ट (□□□.□□□) में समान कार्यान्वयन लागू करना व्यावहारिक नहीं है, क्योंकि इसके सीमित स्क्रिप्टिंग क्षमताओं के कारण। □□□-□□□□ या □□□□□□□□□□ का उपयोग करने की सिफारिश की जाती है।

अतिरिक्त विचार

- **कमांड पार्सिंग:**
  - दोनों कार्यान्वयन केवल कमांड का पहला शब्द नेटवर्क कमांडों की सूची के साथ तुलना करते हैं। उदाहरण के लिए, `git clone` ट्रिगर होता है क्योंकि `git` सूची में है।
  - बहु-शब्द कमांड जैसे `bundle exec jekyll` तब ट्रिगर होंगे जब `bundle` सूची में होगा, जो अधिकांश मामलों के लिए पर्याप्त है।
  - आवश्यकता पड़ने पर, आप कोड को सभी शब्दों को कमांड में चेक करने के लिए संशोधित कर सकते हैं, लेकिन यह गलत सकारात्मक परिणामों को ला सकता है और आम तौर पर अनावश्यक है।
- **प्रॉक्सी चर:**

- दोनों कार्यान्वयन `HTTP_PROXY`, `http_proxy`, `HTTPS_PROXY`, `https_proxy`, `ALL_PROXY`, और `all_proxy` को चेक करते हैं ताकि सामान्य परिवर्तनों को कवर किया जा सके।
- □□□□□□□□ में पर्यावरण चर केस-इन्सेंसिटिव हैं, लेकिन हम दोनों केसों को चेक करते हैं ताकि □□□□-आधारित व्यवहार के साथ सुसंगत रहें (विशेष रूप से □□□-□□□□ में).

□ □□□ **प्रॉक्सी सेटिंग्स:**

- दोनों शेल में `checkproxy` फंक्शन `git config --get` का उपयोग करके □□□-वशिष्ट प्रॉक्सी सेटिंग्स को दिखाता है।

टेस्टिंग

□ **प्रॉक्सी चर सेट करें:**

- □□□-□□□□ में: `export HTTP_PROXY=http://proxy.example.com:8080`
- □□□□□□□□□□ में: `$env:HTTP_PROXY = "http://proxy.example.com:8080"`

□ **नेटवर्क कमांड चलाएं:**

- कमांड जैसे `git --version`, `curl -V` आदि चलाएं।
- प्रॉक्सी सेटिंग्स कमांड के आउटपुट से पहले दिखाए जाएंगे।

□ `checkproxy` **का उपयोग करें:**

- किसी भी शेल में `checkproxy` चलाएं ताकि प्रॉक्सी सेटिंग्स को मैनुअल रूप से देखें।

यह कार्यान्वयन □□□-□□□□ और □□□□□□□□□□ में एक मजबूत प्रॉक्सी चेक प्रदान करता है, जो □□□□□□□ टर्मिनल या स्टैंडअलोन के भीतर उपयोग करने के लिए उपयुक्त है।