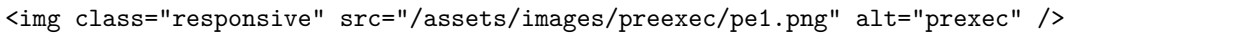
प्रॉक्सी सेटिंग्स प्रदर्शित करने के लिए, आप निम्नलिखित कमांड का उपयोग कर सकते



चीन में रहना या ऐसी कंपनियों में काम करना जो □□□ और प्रॉक्सी का उपयोग करती हैं, सॉफ़्टवेयर डेवलपमेंट को जटिल बना सकता है। इन सेटिंग्स को कॉन्फ़िगर करना भूल जाने से अक्सर कनेक्टिविटी समस्याएं होती हैं। अपने वर्कफ़्लो को सुव्यवस्थित करने के लिए, मैंने □□□□□□□ की मदद से एक सरल □□□ स्क्रिप्ट बनाई है जो आपके प्रॉक्सी सेटिंग्स को स्वचालित रूप से प्रदर्शित करती है जब आप विशिष्ट नेटवर्क-निर्भर कमांड चलाते हैं।

प्रॉक्सी सेटिंग्स क्यों दिखाएं?

प्रॉक्सी और □□□ बाहरी संसाधनों तक सुरक्षित रूप से पहुंचने के लिए आवश्यक हैं। नेटवर्क-निर्भर कमांड्स को निष्पादित करने से पहले अपनी प्रॉक्सी सेटिंग्स दिखाने से आपको कनेक्टिविटी समस्याओं को जल्दी से पहचानने और समस्या निवारण करने में मदद मिलती है।

स्क्रिप्ट

यह स्क्रिप्ट □□□ के `preexec` फ़ंक्शन का उपयोग करती है ताकि यह जांच सके कि आने वाला कमांड नेटवर्क-निर्भर है या नहीं। यदि यह नेटवर्क-निर्भर है और प्रॉक्सी पर्यावरण चर सेट हैं, तो यह वर्तमान प्रॉक्सी सेटिंग्स प्रदर्शित करता है।

```
#
preexec() {
    #    -
    local 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"
        #
    )

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

#
display_proxy() {
    echo -e "\n             :"

    [ -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 ""

#           -
for network_cmd in "${network_commands[@]}"; do
    if [[ "$1" == "$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

}

## Zsh

### 1.    `.zshrc`

                    `.zshrc`                    :

```bash
nano ~/.zshrc
```

2. preexec **फ़ंक्शन जोड़ें**

फ़ाइल के अंत में उपरोक्त स्क्रिप्ट को पेस्ट करें।
```

3. सहेजें और बंद करें

फ़ाइल को सहेजने के लिए `CTRL + O` दबाएं और बाहर निकलने के लिए `CTRL + X` दबाएं।

4. परिवर्तन लागू करें

अपने `.zshrc` को तुरंत नई कॉन्फ़िगरेशन लागू करने के लिए रीलोड करें:

```
source ~/.zshrc
```

सेटअप का परीक्षण

1. प्रॉक्सी सक्षम के साथ

एक प्रॉक्सी वेरिएबल को अस्थायी रूप से सेट करें और `pip` का उपयोग करके एक नेटवर्क-निर्भर कमांड चलाएं:

```
export HTTP_PROXY="http://127.0.0.1:7890"
pip install selenium beautifulsoup4 urllib3
```

(नोट: कोड ब्लॉक को अनुवादित नहीं किया गया है क्योंकि यह प्रोग्रामिंग भाषा में लिखा गया है और इसे बदलने की आवश्यकता नहीं है।)

अपेक्षित आउटपुट:

```
(                     )

             :
   - HTTP_PROXY: http://127.0.0.1:7890
   - http_proxy: 127.0.0.1:7890
   - HTTPS_PROXY: 127.0.0.1:7890
   - https_proxy: 127.0.0.1:7890
   - ALL_PROXY: 127.0.0.1:7890
   - all_proxy: 127.0.0.1:7890


     -4.x.x-py2.py3-none-any.whl (xxx kB)
    4
  beautifulsoup4-4.x.x-py3-none-any.whl (xxx kB)
urllib3
```

```
  urllib3-1.x.x-py2.py3-none-any.whl (xxx kB)
...
```

2. प्रॉक्सी सक्षम किए बिना

प्रॉक्सी वेरिएबल को अनसेट करें और वही `pip` कमांड चलाएं:

```
unset HTTP_PROXY
pip install selenium beautifulsoup4 urllib3
```

(यह कोड ब्लॉक में है, इसलिए इसे अनुवादित नहीं किया गया है।)

अपेक्षित आउटपुट:

```
selenium
  selenium-4.x.x-py2.py3-none-any.whl (xxx kB)
beautifulsoup4
  beautifulsoup4-4.x.x-py3-none-any.whl (xxx kB)
urllib3
  urllib3-1.x.x-py2.py3-none-any.whl (xxx kB)
...
```

(कोई प्रॉक्सी सूचना प्रदर्शित नहीं होनी चाहिए।)

3. नेटवर्क रहित कमांड

स्थानीय कमांड चलाएं जैसे `ls`:

```
ls
```

अपेक्षित आउटपुट:

```
[               ]
```

(कोई प्रॉक्सी सूचना प्रदर्शित नहीं होनी चाहिए।)

अनुकूलन

- network_commands का विस्तार करें: network_commands सरणी में किसी भी अतिरिक्त नेटवर्क-निर्भर कमांड को जोड़ें।
- एलियास को हैंडल करें: सुनिश्चित करें कि नेटवर्क-निर्भर कमांड्स के लिए किसी भी एलियास को network_commands सूची में शामिल किया गया है।

```
alias gpa='git push all'
```

यहां gpa नामक एक एलियास बनाया गया है जो git push all कमांड को संक्षिप्त रूप में प्रदर्शित करता है। इसका उपयोग करके आप git push all कमांड को gpa लिखकर चला सकते हैं।

"gpa" को network_commands सरणी में जोड़ें ताकि इस उपनाम का उपयोग करते समय प्रॉक्सी सूचनाएं ट्रिगर हो सकें।

- रंगों के साथ दृश्यता बढ़ाएं:

बेहतर दृश्यता के लिए, विशेष रूप से अव्यवस्थित टर्मिनल में, आप प्रॉक्सी सूचनाओं में रंग जोड़ सकते हैं:

```
  #    .zshrc
  GREEN='\033[0;32m'
  NC='\033[0m' #

display_proxy() {
  echo -e "\n${GREEN}             :${NC}"
}

[ -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 ""

} “‘
```

निष्कर्ष

प्रॉक्सी सेटिंग्स को प्रबंधित करना प्रतिबंधित नेटवर्क वातावरण में सॉफ्टवेयर विकास को सुचारू रूप से चलाने के लिए महत्वपूर्ण है। यह □□□ स्क्रिप्ट सुनिश्चित करती है कि जब आप नेटवर्क एक्सेस की आवश्यकता वाले कमांड चलाते हैं, तो आप हमेशा अपनी प्रॉक्सी कॉन्फ़िगरेशन के बारे में सूचित रहें, जिससे आपका वर्कफ़्लो बेहतर होता है और समस्या निवारण की दक्षता बढ़ती है।

कोडिंग का आनंद लें! □