Ein Proxy-Check in Windows implementieren

Um einen Proxy-Check in Windows mit Git-Bash und PowerShell zu implementieren, müssen Sie Mechanismen in jeder Shell einrichten, um zu erkennen, wann netzwerkbezogene Befehle ausgeführt werden, und Proxy-Einstellungen anzuzeigen, wenn sie konfiguriert sind. Hier sind die Schritte und der Code für beide Git-Bash und PowerShell. Der "Terminal"bezieht sich wahrscheinlich auf Windows Terminal, das diese Shells hostet, daher konzentrieren wir uns auf die Implementierungen in Git-Bash und PowerShell.

Für Git-Bash

Git-Bash ist eine Bash-Emulation auf Windows, und wir können eine `DEBUG`-Falle verwenden, um eine Funktion vor der Ausführung jedes Befehls auszuführen. Das Ziel ist es, zu überprüfen, ob der Befehl netzwerkbezogen ist und ob Proxy-Einstellungen gesetzt sind, und sie dann anzuzeigen.

Schritte:

1. **Definieren Sie die Liste der netzwerkbezogenen Befehle.**
2. **Erstellen Sie eine Funktion, um Proxy-Einstellungen anzuzeigen.**
3. **Erstellen Sie eine Funktion, um den Befehl und die Proxy-Einstellungen zu überprüfen.**
4. **Richten Sie die `DEBUG`-Falle ein, um die Überprüfung vor jedem Befehl auszuführen.**
5. **Definieren Sie eine manuelle `checkproxy`-Funktion, um Proxy-Einstellungen auf Anfrage anzuzeigen.**
6. **Fügen Sie alle Konfigurationen zur `.bashrc`-Datei hinzu.**

Implementierung: Fügen Sie den folgenden Code zu Ihrer `~/.bashrc`-Datei hinzu (erstellen Sie sie, wenn sie nicht existiert):

```
# Liste der netzwerkbezogenen Befehle
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"
)

# Funktion zum Anzeigen der Proxy-Einstellungen
display_proxy() {
    echo -e "  **Proxy-Einstellungen erkannt:**"
    [ -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 ""
}

# Funktion zum Überprüfen, ob der Befehl netzwerkbezogen ist und Proxys gesetzt sind
proxy_check() {
    local cmd
    # Extrahieren Sie das erste Wort des Befehls
    cmd=$(echo "$BASH_COMMAND" | awk '{print $1}')

    for network_cmd in "${network_commands[@]}"; do
        if [[ "$cmd" == "$network_cmd" ]]; then
            # Überprüfen Sie, ob irgendwelche Proxy-Umgebungsvariablen gesetzt sind
            if [ -n "$HTTP_PROXY" ] || [ -n "$http_proxy" ] || \
               [ -n "$HTTPS_PROXY" ] || [ -n "$https_proxy" ] || \
               [ -n "$ALL_PROXY" ] || [ -n "$all_proxy" ]; then
                display_proxy
            fi
            break
        fi
    done
}

# Setzen Sie die DEBUG-Falle, um proxy_check vor jedem Befehl auszuführen
trap 'proxy_check' DEBUG

# Funktion zum manuellen Überprüfen der Proxy-Einstellungen
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
}
```

Funktionsweise:

- Das `network_commands`-Array listet Befehle auf, die netzwerkbezogen sind.

- `display_proxy` zeigt alle relevanten Proxy-Umgebungsvariablen an, wenn sie gesetzt sind.
- `proxy_check` verwendet `BASH_COMMAND` (in der `DEBUG`-Falle verfügbar), um den auszuführenden Befehl zu erhalten, extrahiert das erste Wort und überprüft, ob es mit einem Netzwerkbefehl übereinstimmt. Wenn Proxy-Variablen gesetzt sind, zeigt es sie an.
- Die Zeile `trap 'proxy_check' DEBUG` stellt sicher, dass `proxy_check` vor jedem Befehl ausgeführt wird.
- `checkproxy` ermöglicht es Ihnen, Proxy-Einstellungen manuell anzuzeigen, einschließlich Git-spezifischer Proxy-Konfigurationen.
- Nach dem Hinzufügen zu `.bashrc` starten Sie Git-Bash neu oder führen `source ~/.bashrc` aus, um die Änderungen zu übernehmen.

Verwendung:

- Wenn Sie einen Netzwerkbefehl ausführen (z. B. `git clone`, `curl`), werden die Proxy-Einstellungen, wenn sie konfiguriert sind, vor der Ausführung des Befehls angezeigt.
- Führen Sie `checkproxy` aus, um Proxy-Einstellungen manuell anzuzeigen.

Für PowerShell

PowerShell hat kein direktes Äquivalent zur Bash-`DEBUG`-Falle, aber wir können den `CommandValidationHandler` des `PSReadLine`-Moduls verwenden, um ähnliche Funktionalität zu erreichen. Dieser Handler wird vor jedem Befehl ausgeführt und ermöglicht es uns, Netzwerkbefehle und Proxy-Einstellungen zu überprüfen.

Schritte:

1. **Definieren Sie die Liste der netzwerkbezogenen Befehle.**
2. **Erstellen Sie eine Funktion, um Proxy-Einstellungen anzuzeigen.**
3. **Richten Sie den `CommandValidationHandler` ein, um Befehle und Proxy-Einstellungen zu überprüfen.**
4. **Definieren Sie eine manuelle `checkproxy`-Funktion, um Proxy-Einstellungen auf Anfrage anzuzeigen.**
5. **Fügen Sie alle Konfigurationen zu Ihrem PowerShell-Profil hinzu.**

Implementierung: Lokalisieren Sie Ihre PowerShell-Profildatei, indem Sie `$PROFILE` in PowerShell ausführen. Wenn sie nicht existiert, erstellen Sie sie:

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

Fügen Sie den folgenden Code zu Ihrem PowerShell-Profil (z. B. `Microsoft.PowerShell_profile.ps1`) hinzu:

```
# Liste der netzwerkbezogenen Befehle
$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"
)

# Funktion zum Anzeigen der Proxy-Einstellungen
function Display-Proxy {
    Write-Host "  **Proxy-Einstellungen erkannt:**"
    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 ""
}

# Richten Sie den CommandValidationHandler ein, um Befehle vor der Ausführung zu überprüfen
Set-PSReadLineOption -CommandValidationHandler {
    param($command)
    # Extrahieren Sie das erste Wort des Befehls
    $cmd = ($command -split ' ')[0]

    if ($networkCommands -contains $cmd) {
        # Überprüfen Sie, ob irgendwelche Proxy-Umgebungsvariablen gesetzt sind
        if ($env:HTTP_PROXY -or $env:http_proxy -or $env:HTTPS_PROXY -or $env:https_proxy -or $env:ALL_PROXY -
            Display-Proxy
        }
    }
    # Stellen Sie sicher, dass der Befehl ausgeführt wird
    return $true
}

# Funktion zum manuellen Überprüfen der Proxy-Einstellungen
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
```

```
}
```

Funktionsweise:

- `$networkCommands` ist ein Array von netzwerkbezogenen Befehlen.
- `Display-Proxy` zeigt alle relevanten Proxy-Umgebungsvariablen an, wenn sie gesetzt sind.
- `Set-PSReadLineOption -CommandValidationHandler` definiert einen Skriptblock, der vor jedem Befehl ausgeführt wird:
  - Es extrahiert das erste Wort des Befehls.
  - Überprüft, ob es in `$networkCommands` enthalten ist.
  - Wenn Proxy-Variablen gesetzt sind, ruft es `Display-Proxy` auf.
  - Gibt `$true` zurück, um sicherzustellen, dass der Befehl ausgeführt wird.
- `checkproxy` ermöglicht das manuelle Anzeigen von Proxy-Einstellungen, einschließlich Git-spezifischer Proxys.
- Nach dem Hinzufügen zu Ihrem Profil starten Sie PowerShell neu oder führen `. $PROFILE` aus, um die Änderungen zu übernehmen.

Anforderungen:

- Das `PSReadLine`-Modul ist erforderlich, das standardmäßig in PowerShell 5.1 und später enthalten ist.
- Wenn eine ältere Version verwendet wird, muss PowerShell möglicherweise aktualisiert oder eine alternative Methode gefunden werden (nicht hier abgedeckt, da die meisten Systeme neuere Versionen verwenden).

Verwendung:

- Wenn Sie einen Netzwerkbefehl ausführen (z. B. `git pull`, `curl`), werden die Proxy-Einstellungen, wenn sie konfiguriert sind, vor der Ausführung des Befehls angezeigt.
- Führen Sie `checkproxy` aus, um Proxy-Einstellungen manuell anzuzeigen.

Hinweise zum “Terminal”

- Wenn “Terminal”sich auf Windows Terminal bezieht, handelt es sich einfach um einen Host für Shells wie Git-Bash, PowerShell oder Command Prompt (cmd.exe).
- Die oben genannten Implementierungen funktionieren innerhalb von Git-Bash- oder PowerShell-Sitzungen in Windows Terminal.

- Die Implementierung ähnlicher Funktionalität in Command Prompt (cmd.exe) ist aufgrund seiner begrenzten Skriptfähigkeiten nicht praktikabel. Es wird empfohlen, Git-Bash oder PowerShell zu verwenden.

Zusätzliche Überlegungen

- **Befehlsanalyse:**
  - Beide Implementierungen überprüfen nur das erste Wort des Befehls gegen die Liste der Netzwerkbefehle. Zum Beispiel löst `git clone` aus, weil `git` in der Liste ist.
  - Mehrwortbefehle wie `bundle exec jekyll` lösen aus, wenn `bundle` in der Liste ist, was für die meisten Fälle ausreicht.
  - Wenn nötig, könnte der Code so geändert werden, dass alle Wörter im Befehl überprüft werden, aber dies könnte zu falschen Positiven führen und ist in der Regel unnötig.
- **Proxy-Variablen:**
  - Beide Implementierungen überprüfen `HTTP_PROXY`, `http_proxy`, `HTTPS_PROXY`, `https_proxy`, `ALL_PROXY` und `all_proxy`, um gängige Variationen abzudecken.
  - In Windows sind Umgebungsvariablen groß- und kleinschreibungsunabhängig, aber wir überprüfen beide Fälle zur Konsistenz mit Unix-ähnlichem Verhalten (insbesondere in Git-Bash).
- **Git-Proxy-Einstellungen:**
  - Die `checkproxy`-Funktion in beiden Shells zeigt Git-spezifische Proxy-Einstellungen mit `git config --get` an.

Testen

- **Proxy-Variablen setzen:**
  - In Git-Bash: `export HTTP_PROXY=http://proxy.example.com:8080`
  - In PowerShell: `$env:HTTP_PROXY = "http://proxy.example.com:8080"`
- **Netzwerkbefehle ausführen:**
  - Versuchen Sie Befehle wie `git --version`, `curl -V` usw.
  - Proxy-Einstellungen sollten vor der Befehlsausgabe angezeigt werden.
- **Verwenden Sie `checkproxy`:**
  - Führen Sie `checkproxy` in jeder Shell aus, um Proxy-Einstellungen manuell anzuzeigen.

Diese Implementierung bietet einen robusten Proxy-Check in beiden Git-Bash und PowerShell, geeignet für die Verwendung innerhalb von Windows Terminal oder eigenständig.