Mise en œuvre d'un contrôle de proxy sous Windows

Pour mettre en œuvre un contrôle de proxy sous Windows en utilisant Git-Bash et PowerShell, il est nécessaire de configurer des mécanismes dans chaque shell pour détecter l'exécution de commandes liées au réseau et afficher les paramètres de proxy s'ils sont configurés. Voici les étapes et le code pour Git-Bash et PowerShell. Le "terminal"mentionné fait probablement référence à Windows Terminal hébergeant ces shells, nous nous concentrerons donc sur les implémentations Git-Bash et PowerShell.

Pour Git-Bash

Git-Bash est une émulation Bash sur Windows, et nous pouvons utiliser un piège `DEBUG` pour exécuter une fonction avant l'exécution de chaque commande. L'objectif est de vérifier si la commande est liée au réseau et si les paramètres de proxy sont définis, puis de les afficher.

Étapes :

1. **Définir la liste des commandes liées au réseau.**
2. **Créer une fonction pour afficher les paramètres de proxy.**
3. **Créer une fonction pour vérifier la commande et les paramètres de proxy.**
4. **Configurer le piège `DEBUG` pour exécuter la vérification avant chaque commande.**
5. **Définir une fonction manuelle `checkproxy` pour afficher les paramètres de proxy à la demande.**
6. **Ajouter toutes les configurations au fichier `.bashrc`.**

Implémentation : Ajoutez le code suivant à votre fichier `~/.bashrc` (créez-le s'il n'existe pas) :

```
# Liste des commandes liées au réseau
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"
)

# Fonction pour afficher les paramètres de proxy
display_proxy() {
    echo -e "  **Paramètres de Proxy Détectés :**"
    [ -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 ""
}

# Fonction pour vérifier si la commande est liée au réseau et si les proxies sont définis
proxy_check() {
    local cmd
    # Extraire le premier mot de la commande
    cmd=$(echo "$BASH_COMMAND" | awk '{print $1}')

    for network_cmd in "${network_commands[@]}"; do
        if [[ "$cmd" == "$network_cmd" ]]; then
            # Vérifier si une des variables d'environnement de proxy est définie
            if [ -n "$HTTP_PROXY" ] || [ -n "$http_proxy" ] || \
               [ -n "$HTTPS_PROXY" ] || [ -n "$https_proxy" ] || \
               [ -n "$ALL_PROXY" ] || [ -n "$all_proxy" ]; then
                display_proxy
            fi
            break
        fi
    done
}

# Définir le piège DEBUG pour exécuter proxy_check avant chaque commande
trap 'proxy_check' DEBUG

# Fonction pour vérifier manuellement les paramètres de proxy
checkproxy() {
    echo "HTTP_PROXY: $HTTP_PROXY"
    echo "HTTPS_PROXY: $HTTPS_PROXY"
    echo "Proxy HTTP Git :"
    git config --get http.proxy
    echo "Proxy HTTPS Git :"
    git config --get https.proxy
}
```

Comment ça fonctionne :

- Le tableau `network_commands` liste les commandes liées au réseau.
- `display_proxy` affiche toutes les variables d'environnement de proxy pertinentes si elles sont définies.
- `proxy_check` utilise `BASH_COMMAND` (disponible dans le piège `DEBUG`) pour obtenir la commande en cours d'

exécution, extrait le premier mot et vérifie s'il correspond à une commande réseau. Si les variables de proxy sont définies, il les affiche.

- La ligne `trap 'proxy_check' DEBUG` assure que `proxy_check` s'exécute avant chaque commande.
- `checkproxy` permet de visualiser manuellement les paramètres de proxy, y compris les configurations de proxy spécifiques à Git.
- Après avoir ajouté ceci à `.bashrc`, redémarrez Git-Bash ou exécutez `source ~/.bashrc` pour appliquer les modifications.

Utilisation :

- Lorsque vous exécutez une commande réseau (par exemple, `git clone`, `curl`), si les paramètres de proxy sont configurés, ils seront affichés avant l'exécution de la commande.
- Exécutez `checkproxy` pour visualiser manuellement les paramètres de proxy.

Pour PowerShell

PowerShell n'a pas d'équivalent direct au piège `DEBUG` de Bash, mais nous pouvons utiliser le gestionnaire `CommandValidationHandler` du module `PSReadLine` pour obtenir une fonctionnalité similaire. Ce gestionnaire s'exécute avant chaque commande, nous permettant de vérifier les commandes réseau et les paramètres de proxy.

Étapes :

1. **Définir la liste des commandes liées au réseau.**
2. **Créer une fonction pour afficher les paramètres de proxy.**
3. **Configurer le gestionnaire `CommandValidationHandler` pour vérifier les commandes et les paramètres de proxy.**
4. **Définir une fonction manuelle `checkproxy` pour afficher les paramètres de proxy à la demande.**
5. **Ajouter toutes les configurations à votre profil PowerShell.**

Implémentation : Tout d'abord, localisez votre fichier de profil PowerShell en exécutant `$PROFILE` dans PowerShell. Si elle n'existe pas, créez-la :

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

Ajoutez le code suivant à votre profil PowerShell (par exemple, `Microsoft.PowerShell_profile.ps1`) :

```
# Liste des commandes liées au réseau
$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"
)

# Fonction pour afficher les paramètres de proxy
function Display-Proxy {
    Write-Host "  **Paramètres de Proxy Détectés :**"
    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 ""
}

# Configurer le gestionnaire CommandValidationHandler pour vérifier les commandes avant l'exécution
Set-PSReadLineOption -CommandValidationHandler {
    param($command)
    # Extraire le premier mot de la commande
    $cmd = ($command -split ' ')[0]

    if ($networkCommands -contains $cmd) {
        # Vérifier si une des variables d'environnement de proxy est définie
        if ($env:HTTP_PROXY -or $env:http_proxy -or $env:HTTPS_PROXY -or $env:https_proxy -or $env:ALL_PROXY -
            Display-Proxy
        }
    }
    # Toujours retourner true pour permettre l'exécution de la commande
    return $true
}

# Fonction pour vérifier manuellement les paramètres de proxy
function checkproxy {
    Write-Host "HTTP_PROXY: $env:HTTP_PROXY"
    Write-Host "HTTPS_PROXY: $env:HTTPS_PROXY"
    Write-Host "Proxy HTTP Git :"
    git config --get http.proxy
    Write-Host "Proxy HTTPS Git :"
    git config --get https.proxy
```

```
}
```

Comment ça fonctionne :

- `$networkCommands` est un tableau de commandes liées au réseau.
- `Display-Proxy` affiche toutes les variables d'environnement de proxy pertinentes si elles sont définies.
- `Set-PSReadLineOption -CommandValidationHandler` définit un bloc de script qui s'exécute avant chaque commande :
  - Il extrait le premier mot de la commande.
  - Vérifie s'il est dans `$networkCommands`.
  - Si les variables de proxy sont définies, il appelle `Display-Proxy`.
  - Retourne `$true` pour s'assurer que la commande s'exécute.
- `checkproxy` permet de visualiser manuellement les paramètres de proxy, y compris les proxies spécifiques à Git.
- Après avoir ajouté ceci à votre profil, redémarrez PowerShell ou exécutez `. $PROFILE` pour appliquer les modifications.

Prérequis :

- Le module `PSReadLine` est requis, qui est inclus par défaut dans PowerShell 5.1 et les versions ultérieures.
- Si vous utilisez une version antérieure, vous devrez peut-être mettre à jour PowerShell ou trouver une méthode alternative (non couverte ici, car la plupart des systèmes utilisent des versions plus récentes).

Utilisation :

- Lorsque vous exécutez une commande réseau (par exemple, `git pull`, `curl`), si les paramètres de proxy sont configurés, ils seront affichés avant l'exécution de la commande.
- Exécutez `checkproxy` pour visualiser manuellement les paramètres de proxy.

Notes sur le "Terminal"

- Si "terminal"fait référence à Windows Terminal, il s'agit simplement d'un hôte pour des shells comme Git-Bash, PowerShell ou Command Prompt (cmd.exe).
- Les implémentations ci-dessus fonctionnent dans des sessions Git-Bash ou PowerShell dans Windows Terminal.

- Mettre en œuvre une fonctionnalité similaire dans Command Prompt (cmd.exe) n'est pas pratique en raison de ses capacités de script limitées. Il est recommandé d'utiliser Git-Bash ou PowerShell à la place.

Considérations supplémentaires

- **Analyse des commandes :**
  - Les deux implémentations vérifient uniquement le premier mot de la commande par rapport à la liste des commandes réseau. Par exemple, `git clone` déclenche car `git` est dans la liste.
  - Les commandes à plusieurs mots comme `bundle exec jekyll` déclencheront si `bundle` est dans la liste, ce qui est suffisant pour la plupart des cas.
  - Si nécessaire, vous pourriez modifier le code pour vérifier tous les mots dans la commande, mais cela pourrait entraîner des faux positifs et est généralement inutile.
- **Variables de proxy :**
  - Les deux implémentations vérifient `HTTP_PROXY`, `http_proxy`, `HTTPS_PROXY`, `https_proxy`, `ALL_PROXY` et `all_proxy` pour couvrir les variations courantes.
  - Sous Windows, les variables d'environnement ne sont pas sensibles à la casse, mais nous vérifions les deux cas pour la cohérence avec le comportement Unix-like (surtout dans Git-Bash).
- **Paramètres de proxy Git :**
  - La fonction `checkproxy` dans les deux shells affiche les paramètres de proxy spécifiques à Git en utilisant `git config --get`.

Tests

- **Définir les variables de proxy :**
  - Dans Git-Bash : `export HTTP_PROXY=http://proxy.example.com:8080`
  - Dans PowerShell : `$env:HTTP_PROXY = "http://proxy.example.com:8080"`
- **Exécuter des commandes réseau :**
  - Essayez des commandes comme `git --version`, `curl -V`, etc.
  - Les paramètres de proxy devraient s'afficher avant la sortie de la commande.
- **Utiliser `checkproxy` :**
  - Exécutez `checkproxy` dans l'un ou l'autre shell pour visualiser manuellement les paramètres de proxy.

Cette implémentation fournit un contrôle de proxy robuste dans Git-Bash et PowerShell, adapté à une utilisation dans Windows Terminal ou en standalone.