實施 Windows 代理檢查

在 Windows 中使用 Git-Bash 和 PowerShell 實現代理檢查，需要在每個 shell 中設置機制來檢測網絡相關命令的執行，並顯示代理設置（如果已配置）。以下是 Git-Bash 和 PowerShell 的步驟和代碼。所提到的“終端”可能指的是 Windows Terminal 承載這些 shell，因此我們將專注於 Git-Bash 和 PowerShell 的實現。

Git-Bash

Git-Bash 是 Windows 上的 Bash 模擬，我們可以使用 `DEBUG` 陷阱來在每次命令執行之前運行一個函數。目的是檢查命令是否與網絡相關，並檢查是否設置了代理，然後顯示它們。

步驟：

1. **定義網絡相關命令的列表。**
2. **創建一個函數來顯示代理設置。**
3. **創建一個函數來檢查命令和代理設置。**
4. **設置 `DEBUG` 陷阱在每次命令之前運行檢查。**
5. **定義一個手動的 `checkproxy` 函數來按需顯示代理設置。**
6. **將所有配置添加到 `.bashrc` 文件中。**

實現： 將以下代碼添加到您的 `~/.bashrc` 文件中（如果不存在則創建它）：

```
# 網絡相關命令列表
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"
)

# 函數顯示代理設置
display_proxy() {
    echo -e "  **Proxy Settings Detected:**"
    [ -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 ""
}
```

```
# 函數檢查命令是否與網絡相關並設置代理
proxy_check() {
    local cmd
    # 提取命令的第一個單詞
    cmd=$(echo "$BASH_COMMAND" | awk '{print $1}')

    for network_cmd in "${network_commands[@]}"; do
        if [[ "$cmd" == "$network_cmd" ]]; then
            # 檢查是否設置了任何代理環境變量
            if [ -n "$HTTP_PROXY" ] || [ -n "$http_proxy" ] || \
               [ -n "$HTTPS_PROXY" ] || [ -n "$https_proxy" ] || \
               [ -n "$ALL_PROXY" ] || [ -n "$all_proxy" ]; then
                display_proxy
            fi
            break
        fi
    done
}

# 設置 DEBUG 陷阱在每次命令之前運行 proxy_check
trap 'proxy_check' DEBUG

# 函數手動檢查代理設置
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
}
```

工作原理：

- `network_commands` 數組列出與網絡相關的命令。
- `display_proxy` 顯示所有相關的代理環境變量（如果設置）。
- `proxy_check` 使用 `BASH_COMMAND`（在 `DEBUG` 陷阱中可用）獲取正在執行的命令，提取第一個單詞，並檢查它是否與網絡命令匹配。如果設置了代理變量，則顯示它們。
- `trap 'proxy_check' DEBUG` 行確保在每次命令之前運行 `proxy_check`。
- `checkproxy` 允許您手動查看代理設置，包括 Git 特定的代理配置。

- 添加到 `.bashrc` 後，重新啟動 Git-Bash 或運行 `source ~/.bashrc` 以應用更改。

使用：

- 當您運行網絡命令（例如 `git clone`、`curl`）時，如果配置了代理設置，則在命令執行之前顯示它們。
- 運行 `checkproxy` 以手動查看代理設置。

PowerShell

PowerShell 沒有直接等同於 Bash 的 `DEBUG` 陷阱，但我們可以使用 `PSReadLine` 模塊的 `CommandValidationHandler` 來實現類似的功能。該處理程序在每次命令之前運行，允許我們檢查網絡命令和代理設置。

步驟：

1. **定義網絡相關命令的列表。**
2. **創建一個函數來顯示代理設置。**
3. **設置 `CommandValidationHandler` 來檢查命令和代理設置。**
4. **定義一個手動的 `checkproxy` 函數來按需顯示代理設置。**
5. **將所有配置添加到您的 PowerShell 配置文件中。**

實現： 首先，通過在 PowerShell 中運行 `$PROFILE` 來定位您的 PowerShell 配置文件。如果不存在，請創建它：

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

將以下代碼添加到您的 PowerShell 配置文件（例如 `Microsoft.PowerShell_profile.ps1`）：

```
# 網絡相關命令列表
$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"
)

# 函數顯示代理設置
function Display-Proxy {
    Write-Host "  **Proxy Settings Detected:**"
    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 ""
}

# 設置 CommandValidationHandler 在命令執行之前檢查命令
Set-PSReadLineOption -CommandValidationHandler {
    param($command)
    # 提取命令的第一個單詞
    $cmd = ($command -split ' ')[0]

    if ($networkCommands -contains $cmd) {
        # 檢查是否設置了任何代理環境變量
        if ($env:HTTP_PROXY -or $env:http_proxy -or $env:HTTPS_PROXY -or $env:https_proxy -or $env:ALL_PROXY -
            Display-Proxy
        }
    }
    # 總是返回 true 以允許命令執行
    return $true
}

# 函數手動檢查代理設置
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
}
```

工作原理：

- `$networkCommands` 是網絡相關命令的數組。
- `Display-Proxy` 顯示所有相關的代理環境變量（如果設置）。
- `Set-PSReadLineOption -CommandValidationHandler` 定義一個在每次命令之前運行的腳本塊：
  - 它提取命令的第一個單詞。
  - 檢查它是否在 `$networkCommands` 中。
  - 如果設置了代理變量，則調用 `Display-Proxy`。
  - 返回 `$true` 以確保命令執行。
- `checkproxy` 允許手動查看代理設置，包括 Git 特定的代理。

- 添加到配置文件後，重新啟動 PowerShell 或運行 `. $PROFILE` 以應用更改。

要求：

- 需要 `PSReadLine` 模塊，該模塊在 PowerShell 5.1 及更高版本中默認包含。
- 如果使用較舊版本，可能需要升級 PowerShell 或找到替代方法（這裡不涵蓋，因為大多數系統使用較新版本）。

使用：

- 當您運行網絡命令（例如 `git pull`、`curl`）時，如果配置了代理設置，則在命令執行之前顯示它們。
- 運行 `checkproxy` 以手動查看代理設置。

關於“終端”

- 如果“終端”指的是 Windows Terminal，它只是 Git-Bash、PowerShell 或命令提示符（cmd.exe）的宿主。
- 上述實現在 Windows Terminal 中的 Git-Bash 或 PowerShell 會話中工作。
- 在命令提示符（cmd.exe）中實現類似功能不切實際，因為其腳本功能有限。建議使用 Git-Bash 或 PowerShell。

其他考慮

- **命令解析：**
  - 兩種實現都僅檢查命令的第一個單詞與網絡命令列表的匹配。例如，`git clone` 會觸發，因為 `git` 在列表中。
  - 多單詞命令如 `bundle exec jekyll` 會觸發，如果 `bundle` 在列表中，這對大多數情況來說是足夠的。
  - 如果需要，可以修改代碼以檢查命令中的所有單詞，但這可能會導致假陽性並通常是不必要的。
- **代理變量：**
  - 兩種實現都檢查 `HTTP_PROXY`、`http_proxy`、`HTTPS_PROXY`、`https_proxy`、`ALL_PROXY` 和 `all_proxy` 以涵蓋常見變體。
  - 在 Windows 中，環境變量不區分大小寫，但我們檢查兩種情況以與 Unix 類似的行為一致（特別是在 Git-Bash 中）。
- **Git 代理設置：**
  - 兩個 shell 中的 `checkproxy` 函數使用 `git config --get` 顯示 Git 特定的代理設置。

測試

- **設置代理變量：**
  - 在 Git-Bash 中：`export HTTP_PROXY=http://proxy.example.com:8080`
  - 在 PowerShell 中：`$env:HTTP_PROXY = "http://proxy.example.com:8080"`
- **運行網絡命令：**
  - 試試命令如 `git --version`、`curl -V` 等。
  - 代理設置應在命令輸出之前顯示。
- **使用 `checkproxy`：**
  - 在任何 shell 中運行 `checkproxy` 以手動查看代理設置。

這種實現在 Git-Bash 和 PowerShell 中提供了強大的代理檢查，適合在 Windows Terminal 或獨立使用。