Implementing a Proxy Check in Windows

To implement a proxy check in Windows using Git-Bash and PowerShell, you need to set up mechanisms in each shell to detect when network-related commands are executed and display proxy settings if they are configured. Below are the steps and code for both Git-Bash and PowerShell. The "terminal"mentioned likely refers to Windows Terminal hosting these shells, so we'll focus on Git-Bash and PowerShell implementations.

For Git-Bash

Git-Bash is a Bash emulation on Windows, and we can use a `DEBUG` trap to run a function before each command execution. The goal is to check if the command is network-related and if proxy settings are set, then display them.

Steps:

1. **Define the list of network-related commands.**
2. **Create a function to display proxy settings.**
3. **Create a function to check the command and proxy settings.**
4. **Set up the `DEBUG` trap to run the check before each command.**
5. **Define a manual `checkproxy` function to display proxy settings on demand.**
6. **Add all configurations to the `.bashrc` file.**

Implementation: Add the following code to your `~/.bashrc` file (create it if it doesn't exist):

```
# List of network-related commands
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"
)

# Function to display proxy settings
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 ""
```

```
}

# Function to check if the command is network-related and proxies are set
proxy_check() {
    local cmd
    # Extract the first word of the command
    cmd=$(echo "$BASH_COMMAND" | awk '{print $1}')

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

# Set the DEBUG trap to run proxy_check before each command
trap 'proxy_check' DEBUG

# Function to manually check proxy settings
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
}
```

How It Works:

- The `network_commands` array lists commands that are network-related.
- `display_proxy` shows all relevant proxy environment variables if they are set.
- `proxy_check` uses `BASH_COMMAND` (available in the `DEBUG` trap) to get the command being executed, extracts the first word, and checks if it matches any network command. If proxy variables are set, it displays

them.

- The `trap 'proxy_check' DEBUG` line ensures `proxy_check` runs before each command.
- `checkproxy` allows you to manually view proxy settings, including Git-specific proxy configurations.
- After adding this to `.bashrc`, restart Git-Bash or run `source ~/.bashrc` to apply changes.

Usage:

- When you run a network command (e.g., `git clone`, `curl`), if proxy settings are configured, they will be displayed before the command executes.
- Run `checkproxy` to manually view proxy settings.

For PowerShell

PowerShell doesn't have a direct equivalent to Bash's `DEBUG` trap, but we can use the `PSReadLine` module's `CommandValidationHandler` to achieve similar functionality. This handler runs before each command, allowing us to check for network commands and proxy settings.

Steps:

1. **Define the list of network-related commands.**
2. **Create a function to display proxy settings.**
3. **Set up the `CommandValidationHandler` to check commands and proxy settings.**
4. **Define a manual `checkproxy` function to display proxy settings on demand.**
5. **Add all configurations to your PowerShell profile.**

Implementation: First, locate your PowerShell profile file by running `$PROFILE` in PowerShell. If it doesn't exist, create it:

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

Add the following code to your PowerShell profile (e.g., `Microsoft.PowerShell_profile.ps1`):

```
# List of network-related commands
$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 to display proxy settings
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 ""
}

# Set up the CommandValidationHandler to check commands before execution
Set-PSReadLineOption -CommandValidationHandler {
    param($command)
    # Extract the first word of the command
    $cmd = ($command -split ' ')[0]

    if ($networkCommands -contains $cmd) {
        # Check if any proxy environment variables are set
        if ($env:HTTP_PROXY -or $env:http_proxy -or $env:HTTPS_PROXY -or $env:https_proxy -or $env:ALL_PROXY -
            Display-Proxy
        }
    }
    # Always return true to allow command execution
    return $true
}

# Function to manually check proxy settings
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
}
```

How It Works:

- `$networkCommands` is an array of network-related commands.
- `Display-Proxy` displays all relevant proxy environment variables if they are set.

- `Set-PSReadLineOption -CommandValidationHandler` defines a script block that runs before each command:
  - It extracts the first word of the command.
  - Checks if it's in `$networkCommands`.
  - If proxy variables are set, it calls `Display-Proxy`.
  - Returns `$true` to ensure the command executes.
- `checkproxy` allows manual viewing of proxy settings, including Git-specific proxies.
- After adding to your profile, restart PowerShell or run `. $PROFILE` to apply changes.

Requirements:

- The `PSReadLine` module is required, which is included by default in PowerShell 5.1 and later.
- If using an older version, you may need to upgrade PowerShell or find an alternative method (not covered here, as most systems use newer versions).

Usage:

- When you run a network command (e.g., `git pull`, `curl`), if proxy settings are configured, they will be displayed before the command executes.
- Run `checkproxy` to manually view proxy settings.

Notes on the "Terminal"

- If "terminal"refers to Windows Terminal, it is simply a host for shells like Git-Bash, PowerShell, or Command Prompt (cmd.exe).
- The above implementations work within Git-Bash or PowerShell sessions in Windows Terminal.
- Implementing similar functionality in Command Prompt (cmd.exe) is not practical due to its limited scripting capabilities. It's recommended to use Git-Bash or PowerShell instead.

Additional Considerations

- **Command Parsing:**
  - Both implementations check only the first word of the command against the network commands list. For example, `git clone` triggers because `git` is in the list.
  - Multi-word commands like `bundle exec jekyll` will trigger if `bundle` is in the list, which is sufficient for most cases.

  - If needed, you could modify the code to check all words in the command, but this might lead to false positives and is generally unnecessary.

- **Proxy Variables:**
  - Both implementations check for `HTTP_PROXY`, `http_proxy`, `HTTPS_PROXY`, `https_proxy`, `ALL_PROXY`, and `all_proxy` to cover common variations.
  - In Windows, environment variables are case-insensitive, but we check both cases for consistency with Unix-like behavior (especially in Git-Bash).
- **Git Proxy Settings:**
  - The `checkproxy` function in both shells displays Git-specific proxy settings using `git config --get`.

Testing

- **Set Proxy Variables:**
  - In Git-Bash: `export HTTP_PROXY=http://proxy.example.com:8080`
  - In PowerShell: `$env:HTTP_PROXY = "http://proxy.example.com:8080"`
- **Run Network Commands:**
  - Try commands like `git --version`, `curl -V`, etc.
  - Proxy settings should display before the command output.
- **Use `checkproxy`:**
  - Run `checkproxy` in either shell to manually view proxy settings.

This implementation provides a robust proxy check in both Git-Bash and PowerShell, suitable for use within Windows Terminal or standalone.