GIT

- Installation and configuration of GIT
- Enterprise workplace GIT structure
- Configure GIT for TLS inspection

Installation and configuration of GIT

To get started with GIT, first install Visual Studio Code (VS Code) and GIT on your computer. After installation, configure GIT with your name and email address. These steps are documented here.

Install VS Code

Install VS Code on your computer. Here is a winget command for Windows.

```
winget install -e --id Microsoft.VisualStudioCode
```

You can also download it on the official page for other platforms:

Download Visual Studio Code - Mac, Linux, Windows

Install GIT

Install GIT on your computer (Command for Windows only):

```
winget install -e --id Git.Git
```

For other platforms use this download links.

Git - Downloads (git-scm.com)

Configure GIT

Configure GIT on first startup:

1. Name and email: Open your terminal or GIT bash (if using Windows) and configure GIT by providing your name and email address. This helps GIT to identify you as the author of the changes:

```
git config --global user.name "<yourdisplayName>"
git config --global user.email "<yourEmailAdress>"
```

2. Texteditor (optional): If you want, you can select your preferred text editor for GIT. For example, to set Visual Studio Code as the default editor.

```
git config --global core.editor "code --wait"
```

Enterprise workplace GIT structure

In this concept, we establish a central repository for PowerShell scripts to encourage collaboration and active work on scripts. We rely on the following elements:

1. **Central repository for PowerShell Scripts:** We set up a central repository where all PowerShell scripts are stored. This helps to organize and unify script management.
2. **Active work in the repository:** All team members actively work in the central repository. This means that all changes to PowerShell scripts can be tracked, documented and monitored in this repository.
3. **VSCode as a coding environment:** Visual Studio Code (VSCode) can be used as an Integrated Development Environment (IDE) to develop and edit the PowerShell scripts. VSCode provides a wide range of extensions and features that facilitate the development and debugging of PowerShell Scripts. This concept only considers working with VSCode as a text editor.
4. **Shared repository:** The central repository is accessible to the entire team. Here, all team members can access the latest versions of the scripts, submit changes, and track the history of the scripts.
5. **Personal repositories (e.g., powershell-luca):** In addition to the shared repository, team members have the ability to create personal repositories to work on specific tasks or scripts that are not yet ready for the main repository. These personal repositories can later be integrated into the main repository once the work has been completed and reviewed. Own scripts created during working time should also be stored there, so that synergies can be used more often.

Overall, this concept aims to create an efficient and collaborative development environment for PowerShell scripts by providing a clear structure for management and collaboration while maintaining flexibility for individual work.

Folder structure

powershell-<teamname>: Under "powershell-<teamname>" PowerShell scripts are centrally managed, access is available to all members. Technology distinction: For technologies not related to PowerShell (e.g., C#), create separate repositories to ensure clear separation. The subfolders of the shared repository are differentiated by technology.

powershell-<username>: Here we organize personal PowerShell scripts. All scripts and programs created during working hours are placed here and also actively worked on. This

repository is also accessible to all team members.

Permissions for all project members: Permissions are designed so that all members of the DevOps project can access all repositories.

Leaving team members: Even after members leave the organisation, the repositories remain intact. However, productive and shared scripts are moved to the central "powershell-wps" repository to be available quicker to all.

Naming concept personal repository

Here is clarity in the folder structure of a GIT repositories.

The naming is as follows:
<programming language>-<name>

For example:
powershell-luca

Configure GIT for TLS inspection

> Bypassing TLS inspection or disabling SSL verification can expose your system and data to serious security risks, including man-in-the-middle attacks. This article is provided strictly for educational and debugging purposes in trusted, controlled environments. Do not use these settings on production systems or networks you do not fully control.

Bypass TLS Inspection for Git Using GIT CONFIG

In enterprise environments, it’s not uncommon for outbound HTTPS traffic to be intercepted and inspected by network security tools. This process—known as TLS inspection or SSL interception—can cause tools like Git to fail when attempting to communicate with remote repositories over HTTPS. A common error you might see is:

```
fatal: unable to access 'https://github.com/user/repo.git/': SSL certificate problem: unable to get local issuer certificate
```

This happens because the intercepted certificate doesn't match what Git expects from the server, especially when a custom or self-signed certificate is involved.

Quick and Dirty: Disable SSL Verification

The fastest way to bypass TLS inspection is to disable SSL verification altogether for Git using this configuration:

```
git config --global http.sslVerify "false"
```

This tells Git not to verify the SSL certificate when connecting to remote repositories over HTTPS.

What This Command Does

- `--global`: Applies the setting for all repositories for the current user.
- `http.sslVerify "false"`: Disables SSL certificate validation.

This bypasses the TLS validation errors caused by mismatched or untrusted certificates during deep packet inspection.

When to Use This

- **Controlled internal environments** where you trust the internal proxy or TLS inspection appliance.
- **Temporary workaround** for debugging connectivity issues.
- **CI/CD environments** behind secure corporate firewalls with certificate inspection.

A Safer Alternative: Add the Root CA to Git's Trust Store

Instead of disabling verification, a better approach is to configure Git to trust the inspection proxy's certificate. Here's how:

1. **Obtain the root certificate** used by the inspection tool (usually a `.crt` file).
2. **Tell Git to use it:**

```
git config --global http.sslCAInfo <pathtocorporate-root-ca>.crt
```

This method allows you to retain SSL verification while trusting your organization's inspection certificate.

Final Notes

- **Never disable SSL verification for external, public repositories unless absolutely necessary.**
- If you use the `http.sslVerify false` setting, consider using it with the `--local` or `--global` scope carefully. You can also override it per repository using: `git config --local http.sslVerify false`
- Always revert to secure practices once the underlying issue (e.g., missing root CA) is resolved.

Method	Security Risk	Use Case
`git config --global http.sslVerify "false"`	High	Temporary debugging in trusted environments
Add custom CA via `http.sslCAInfo`	Low	Permanent, secure solution for TLS interception

When in doubt, favor secure configuration over convenience. If you're working behind a corporate firewall and unsure how to proceed, reach out to your IT department—they may already have a secure solution in place.