[Source: https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html] – *The contents of this source page have been used to create this guidance document.*

All – Below are excerpts from the Getting Started with GitLab guidance page (link above) provided by GitLab. I have eliminated certain sections for clarity and highlighted important steps.

Start using Git on the command line

Git is an open-source distributed version control system designed to handle everything from small to very large projects with speed and efficiency. GitLab is built on top of Git.

While GitLab has a powerful user interface from which you can do a great amount of Git operations directly in the browser, you'll eventually need to use Git through the command line for advanced tasks.

For example, if you need to fix complex merge conflicts, rebase branches, merge manually, or undo and roll back commits, you'll need to use Git from the command line and then push your changes to the remote server.

This guide will help you get started with Git through the command line and can be your reference for Git commands in the future. If you're only looking for a quick reference of Git commands, you can download GitLab's Git Cheat Sheet.

For more information about the advantages of working with Git and GitLab:

- Watch the GitLab Source Code Management Walkthrough video.
- Learn how GitLab became the backbone of Worldline's development environment.

Requirements (make sure these are installed)

You don't need a GitLab account to use Git locally, but for the purpose of this guide we recommend registering and signing into your account before starting. Some commands need a connection between the files in your computer and their version on a remote server.

You'll also need to open a command shell and have Git installed in your computer.

Command shell (Terminal on Mac, Git Bash on Windows)

To execute Git commands in your computer, you'll need to open a command shell (also known as command prompt, terminal, and command line) of your preference. Here are some suggestions:

- For macOS users:
    - Built-in: Terminal. Press `⌘ command` + `space` and type "terminal" to find it.
    - iTerm2, which you can integrate with zsh and oh my zsh for color highlighting, among other handy features for Git users.
- For Windows users:
    - Built-in: **cmd**. Click the search icon on the bottom navbar on Windows and type "cmd" to find it.
    - PowerShell: a Windows "powered up" shell, from which you can execute a greater number of commands.
    - Git Bash: it comes built into Git for Windows.
- For Linux users:
    - Built-in: Linux Terminal.

Install Git

Open a command shell and run the following command to check if Git is already installed in your computer:

```
git --version
```

If you have Git installed, the output will be:

```
git version X.Y.Z
```

If your computer doesn't recognize `git` as a command, you'll need to install Git. After that, run `git --version` again to verify whether it was correctly installed.

Configure Git (This is a best practice, please do this)

To start using Git from your computer, you'll need to enter your credentials (user name and email) to identify you as the author of your work. The user name and email should match the ones you're using on GitLab.

In your shell, add your user name you have for GitLab:

```
git config --global user.name "your_username"
```

And your email address:

```
git config --global user.email "your_email_address@example.com"
```

To check the configuration, run:

```
git config --global --list
```

The `--global` option tells Git to always use this information for anything you do on your system. If you omit `--global` or use `--local`, the configuration will be applied only to the current repository.

You can read more on how Git manages configurations in the Git configuration documentation.

Git authentication methods

To connect your computer with GitLab, you need to add your credentials to identify yourself. You have two options:

- Authenticate on a project-by-project basis through HTTPS, and enter your credentials every time you perform an operation between your computer and GitLab.
- Authenticate through SSH once and GitLab won't ask your credentials every time you pull, push, and clone.

To start the authentication process, we'll clone an existing repository to our computer:

- If you want to use **SSH** to authenticate, follow the instructions on the SSH documentation to set it up before cloning.
- If you want to use **HTTPS**, GitLab will request your user name and password:

    - If you don't have 2FA enabled, use your account's password.
    - If you have 2FA enabled for your account, you'll have to use a Personal Access Token with **read_repository** or **write_repository** permissions instead of your account's password. Create one before cloning.

Authenticating via SSH is GitLab's recommended method. You can read more about credential storage in the Git Credentials documentation.

Clone a repository (Do this for a local copy of the course materials)

To start working locally on an existing remote repository, clone it with the command `git clone <repository path>`. You can either clone it via HTTPS or SSH, according to your preferred authentication method.

You can find both paths (HTTPS and SSH) by navigating to your project's landing page and clicking **Clone**. GitLab will prompt you with both paths, from which you can copy and paste in your command line.

For example, considering our sample project:

- To clone through HTTPS, use `https://gitlab.com/gitlab-tests/sample-project.git`.
- To clone through SSH, use `git@gitlab.com:gitlab-tests/sample-project.git`.

To get started, open a terminal window in the directory you wish to add the repository files into, and run one of the `git clone` commands as described below.

Both commands will download a copy of the files in a folder named after the project's name and preserve the connection with the remote repository. You can then navigate to the new directory with `cd du-den-cyber-pt-12-2020-u-c` and start working on it locally.

Clone via SSH (Recommended method, but not required)

To clone `git@du.bootcampcontent.com:denver-coding-bootcamp/du-den-cyber-pt-12-2020-u-c.git` via SSH:

```
git clone git@du.bootcampcontent.com:denver-coding-bootcamp/du-den-cyber-pt-12-2020-u-c.git
```

Clone via HTTPS

To clone `https://du.bootcampcontent.com/denver-coding-bootcamp/du-den-cyber-pt-12-2020-u-c.git` via HTTPS:

```
git clone https://du.bootcampcontent.com/denver-coding-bootcamp/du-den-
cyber-pt-12-2020-u-c.git
```

On Windows, if you entered incorrect passwords multiple times and GitLab is responding `Access denied`, you may have to add your namespace (user name or group name) to clone through HTTPS: `git clone https://namespace@gitlab.com/gitlab-org/gitlab.git`.

Download the latest changes in the project

To work on an up-to-date copy of the project (it is important to do this every time you start working on a project), you `pull` to get all the changes made by users since the last time you cloned or pulled the project. Use `master` for the `<name-of-branch>` to get the main branch code, or the branch name of the branch you are currently working in (you must run this from the directory that is your cloned repository For most of you, the directory will be ./Downloads/ du-den-cyber-pt-12-2020-u-c where ./Downloads is the directory in your user file structure where you download things. If you use a different directory to run the git clone command from, make sure you do all further git commands while in that folder.

```
git pull
```

When you clone a repository, `REMOTE` is typically `origin`. This is where the repository was cloned from, and it indicates the SSH or HTTPS URL of the repository on the remote server. `<name-of-branch>` is usually `master`, but it may be any existing branch. You can create additional named remotes and branches as necessary.

You can learn more on how Git manages remote repositories in the Git Remote documentation.

Git terminology

If you’re familiar with the Git terminology, you may want to jump directly into the basic commands.

Repository

Your files in GitLab live in a **repository**, similar to how you have them in a folder or directory in your computer. **Remote** repository refers to the files in GitLab and the copy in your computer is called **local** copy. A **project** in GitLab is what holds a repository, which holds your files. Often, the word “repository” is shortened to “repo”.

Download vs clone

To create a copy of a remote repository files on your computer, you can either **download** or **clone** it. If you download it, you cannot sync it with the remote repository on GitLab.

On the other hand, by cloning a repository, you’ll download a copy of its files to your local computer, but preserve the Git connection with the remote repository, so that you can work on the its files on your computer and then upload the changes to GitLab.

Pull and push

After you saved a local copy of a repository and modified its files on your computer, you can upload the changes to GitLab. This is referred to as **pushing** to GitLab, as this is achieved by the command `git push`.

When the remote repository changes, your local copy will be behind it. You can update it with the new changes in the remote repository. This is referred to as **pulling** from GitLab, as this is achieved by the command `git pull`.

Namespace

A **namespace** is either a **user name** or a **group name**.

For example, suppose Jo is a GitLab.com user and they chose their user name as `jo`. You can see Jo’s profile at `https://gitlab.com/jo`. `jo` is a namespace.

Jo also created a group in GitLab, and chose the path `test-group` for their group. The group can be accessed under `https://gitlab.com/test-group`. `test-group` is a namespace.