This document has been elaborated for the project FITPED (https://www.fitped.eu)

FITPED

Work-Based Learning in Future
IT Professionals Education
(Grant. no. 2018-1-SK01-KA203-046382)

Co-funded by the
Erasmus+ Programme
of the European Union

GIT

2020

This project has been funded with support from the European Commission under the ERASMUS+ Programme 2018, KA2, project number: 2018-1-SK01-KA203-046382.

Content

Version Control System

Chapter 1

1.1 Introduction

🕮 1.1.1

Version Control System (VCS) is a software that helps software developers to work together and maintain a complete history of all versions of programming code. VCSs are a category of software tools that help a software team manage changes to source code over time, keep track of every modification to the code in a special kind of database. If a mistake happens, developers can compare earlier versions of the code with the current version to fix the mistake while minimising disruption to all team members.

VCS has the following functions:

- allows developers to work simultaneously,
- does not allow overwriting each other's changes,
- maintains a history of every version.

🕮 1.1.2

Historically, there are two main types of VCS:

- Centralized version control system (CVCS) uses a central server to store all files and enables team collaboration. But the major drawback of CVCS is its single point of failure, i.e., failure of the central server. Unfortunately, if the central server goes down for an hour, then during that hour, no one can collaborate at all. And even in the worst case, if the disk of the central server gets corrupted and proper backup has not been taken, then you will lose the entire history of the project. Here, the distributed version control system (DVCS) comes into the picture.
- Distributed/Decentralized version control system (DVCS) clients not only check out the latest snapshot of the directory but they also fully mirror the repository. If the server goes down, then the repository from any client can be copied back to the server to restore it. Every checkout is a full backup of the repository. Git does not rely on the central server, and that is why you can perform many operations when you are offline. You can commit changes, create branches, view logs, and perform other operations when you are offline. You require a network connection only to publish your changes and take the latest changes.

This tutorial is focused on distributed version control system Git.

📖 1.1.3

The following are the benefits of Version Control Systems:

- track every individual change made by the developer,
- prevent concurrent work from conflicts,
- supports a developer's preferred workflow,
- works on any platform,
- facilitates a smooth and continuous flow of changes to the code rather than the frustrating of file locking,
- stores complete long-term change history of every file,
- allows creating and merging branches,
- allows easily to trace each change made to the software and connect it to project management and bug tracking software.

📖 1.1.4

Git is the most widely used modern VCS. Git is a mature, open-source project with distributed architecture. It belongs to the DVCS. Git has the functionality, performance, security and flexibility that most teams and individual developers need. Git is a de facto standard. Moreover, many third-party software tools and services are already integrated with Git, including IDEs, issue and project tracking software, and code hosting services like GitHub or Bitbucket.

📖 1.1.5

Git comes with several advantages. The most important are as follows:

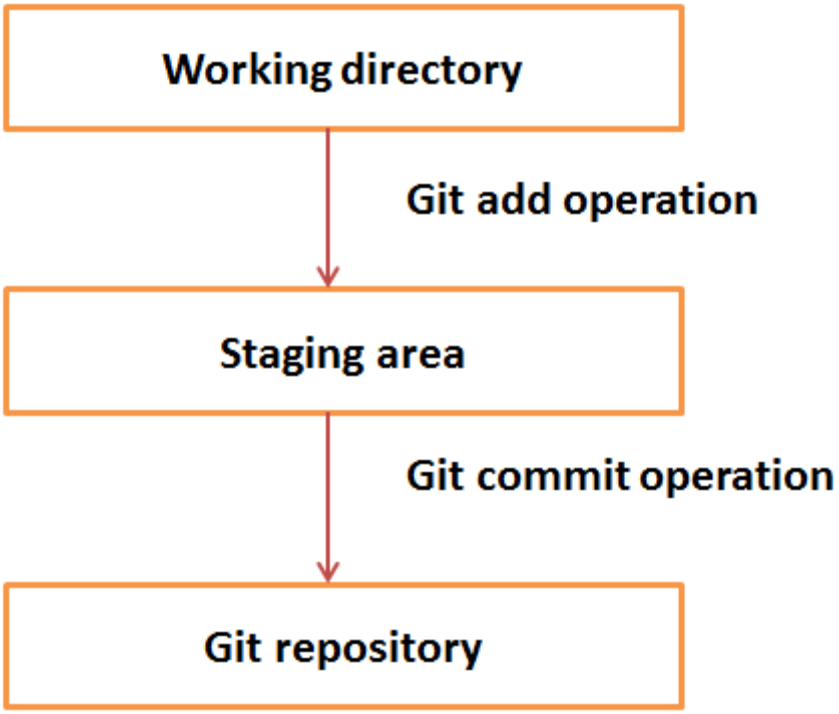
- As was mentioned before, Git belongs to the Free and open-source software. Git is released under GPL's open source license. It is available freely over the internet for download. It is possible to download its source code and perform any changes according to developers' requirements.

- Git is small because it does not require GUI, the core of Git is written in C, and most operations are done locally.
- Git supports implicit backup. It means that the chances of losing data are very rare when there are multiple copies of it. Each client contains the whole repository of source code, which can be used in the event of a crash or disk corruption.
- Because Git uses a cryptographic hash function SHA1 to name and identify objects within its database, Git is considered safe. Each operation is check-summed. Therefore, it is impossible to change file and its properties from the Git database without knowing Git.
- Git does not require powerful hardware because it does not rely on a central server. In the case of Git, similarly to other DVCS, developers do not interact directly with the server unless they need to push or pull changes.
- Using branches require copying, deleting and merging files is complicated and time-consuming. However, branch management with Git is very simple and fast. It takes only a few seconds to create, delete, and merge branches.

🕮 1.1.6

The basic workflow of changes (life cycle) in files consists of the following steps:

1. Files are modified in the working directory.
2. Files are added to the staging area.
3. Operation commit moves the files from the staging area to the local Git repository.



🕮 1.1.7

The file, several files or directories can be in one of the following states, which are available using git command

```
git status
```

- modified – changes have not been yet committed and stored in a local repository,
- staged – modified file or directory are prepared for the next commit,
- committed – changes are stored in a local repository,

🕮 1.1.8

Git can be easily installed on Windows, Linux or Mac OS. There are several ways how to download and install it, for example, from the following official websites (https://gitforwindows.org/, https://git-scm.com/book/en/v2/Getting-Started-Installing-Git.

🕮 1.1.9

Git has several global (system-wide) and local configuration variables, which can be set up using git config tool. Global configurations are stored in the .gitconfig file, which is located in the Git home directory. To set up these configuration variables as global, the --global option must be added. If --global option is omitted, the configuration variables are specific for the current Git repository, as --local option. To set up these values, the user must have root privileges the --system option must be used in commands.

Setting username, which is used by Git for each commit.

```
git config --global user.name <name>
```

Setting email id, which is used by Git for each commit.

```
git config --global user.email <email>
```

Listing current Git settings

To verify your Git settings of the local repository, git config –list command can be used as given below.

```
git config –list
```

It also often necessary to define or change the default text editor used for example by the commit command.

```
git config --system core.editor <editor>
```

where <editor> is a command that launches the desired text editor. The -- system option set the configuration for the entire system, meaning all users and repositories on the computer.

Git Commands

Chapter 2

2.1 Git Commands

🕮 2.1.1

Git contains many commands with a myriad of options and flags, which help a developer to solve many complicated situations. The git help command helps them to find the expected command quickly and provide all required information in a concise manner. This is the simplest version of the command

```
git help <verb>
```

where <verb> represents the search term.

🕮 2.1.2

A new repository can be created using command

```
git init
```

This command creates a new .git subdirectory in the current directory, which is considered as a working directory of the project. Therefore, it is recommended to move to the directory, which should be tracked by the Git, before this initial command is executed. Git can be initialized in existing project directory using the following slightly modified command

🕮 2.1.3

If the central repository has been already created and contains the source code of the project, the command git clone can be used for creating its local working copy. The original repository can be located on the local filesystem or on a remote machine accessible, supported protocols. Once a developer has obtained a working copy, all version control operations are managed through their local repository.

```
git clone <repo url>
```

Repository URL is available in the central repository and will be introduced later. As a convenience, cloning automatically creates a remote connection called "origin" pointing back to the original repository. This makes it very easy to interact with a central repository.

```
git clone <repo url> <directory>
```

this equivalent clones of a remote repository in the defined directory.

📖 2.1.4

In contrast with saving file in the operation system, saving file in Git consists of several steps. A commit command is at the end and therefore is often considered the Git equivalent of a "save". However, in the detailed view, there are three commands: git add, git status, and git commit, which are used in combination to save a snapshot of a Git project's current state.

📖 2.1.5

The *git add* command adds a change in the working directory to the staging area. It tells Git, which updates to a particular file should be included in the next commit. However, *git add* does not really affect the repository in any significant way. Changes are not actually recorded in the local repository until the command git commit runs.

```
git add <file> - stages a particular file to the staging area for the next commit,
git add <directory> - stages all changes in a directory for the next commit,
git add. – adds the whole project directory to the staging area.
```

📖 2.1.6

The following command

```
git status,
```

is often used together with *git add* to view the state of the working directory and the staging area.

It returns the current state of the files in the working directory and staging area.

Other command

```
git log
```

shows detailed information about all the snapshots of the project. For example, it can show the whole history of a selected file

```
git log <file>
```

🕮 2.1.7

The *git commit* command is together with *git add* the most frequent Git command. It captures a snapshot (commits) of the project's currently staged changes and can be thought of as snapshots along the timeline of a Git project. Committed snapshots represent stable, safe versions of a project. Git snapshots are always committed to the local repository. Git does not require to interact with the remote repository until the developer makes this decision. Each developer's local repository, where the commits are stored, is a buffer between the developer's contributions and the central remote repository.

The simplest form of commit command is as follows

```
git commit
```

However, this command launches the text editor and prompt developer for a commit message, which is a mandatory part of each commit. It is often easier for that reason to add the message directly to the commit command using with -m option

```
git commit -m "commit message"
```

If the -a option is included, this combination immediately creates a commit of all the staged changes and takes an inline commit message

```
git commit -am "commit message"
```

🕮 2.1.8

A Git repository can be configured to ignore specific files or directories. This will prevent Git from saving changes to any ignored content stored in files. Git has multiple methods of configuration that manage the ignore list. Ignored files are usually built artefacts and machine-generated files that can be derived from the repository source. Some common examples are:

- dependency caches,

- compiled code, such as .class files
- build output directories, such as /bin, /out,
- files generated at runtime, such as .log, .lock, or .tmp
- hidden system files, such as .DS_Store or Thumbs.db
- personal IDE config files, such as .idea/workspace.xml

Ignored files are tracked in a special file named .gitignore that is checked in at the root of your repository. This file can be created using any text editor, which contains all required git ignore patterns based on the various symbols like **, *, !, and other regular expression wildcards

🕮 2.1.9

Often it is necessary to compare changes between two input files. Git diff command is a multi-use Git command that runs a diff function on Git data sources like commits, branches, files, etc. It is often used along with git status and git log to analyse the current state of a Git repository.

```
git diff
```

This command has many options, which make it a very multi-purpose command. Their detailed description is available on the official documentation site (https://git-scm.com/book/en/v2)

🕮 2.1.10

Sometimes it is necessary to temporarily shelves (stashes) changes made to the working directory and work on something else, and then come back and re-apply them later, without committing. The git command

```
git stash
```

takes staged and unstaged uncommitted changes save them away for later use, and then reverts them from working directory. After it is executed, the developer can make other changes, create new commits, switch branches. Finally, she can come back and re-apply stash with the command

```
git stash pop
```

2.1.11

The abbreviation VCS stands for:

- Version Control System
- Virtual Control System
- Version Combination System
- Very Complicated System
- Value Commited System

2.1.12

Mark the states in which files may be during the Git life cycle.

- staged
- commited
- modified
- aborted
- requested
- submitted

2.1.13

Which of the following Git commands can be used for creating a new repository?

- $ git init
- $ git clone
- $ git copy
- $ git start
- $ git create
- $ git download