Git Notes

Filippo Valmori

9th June 2024

1. Installation

◦ Installation procedure (tested on *Windows 10* OS):

- download and launch installer from Git website (free and open-source);
- set *C:\Program Files\Git* as installation path;
- tick *Open Git Bash here* and untick *Open Git GUI here* within *Windows Explorer integration*;
- select *Use Visual Studio Code as Git's default editor*;
- select *Let Git decide* about default branch naming;
- select *Git from command line and also from 3rd-party software*;
- select *Use bundled OpenSSH*;
- select *Use the OpenSSL library*;
- select *Checkout Windows-style, commit Unix-style line endings*;
- select *Use MinTTY*;
- select *Fast-forward or merge* as pull-command behavior;
- select *Git Credential Manager*;
- tick *Enable file system caching*
- skip the *Experimental options* window and start the installation.

2. Setup

◦ User's name and email configuration:

- to configure Git username → *git config --global user.name "Filippo Valmori"*;
- to configure Git email → *git config --global user.email "filippo.valmori@gmail.com"*;
- **NB #1**: for the last x2 commands, *--system* or *--local* could be used in place of *–global* to (however that's in general not recommended, see Coursera's training for more details);
- to readback set user's name and email → *git config user.name* and git config user.email (or all at once via *git config [--global] --list*);
- to avoid line-ending issues among team members working with different OSs and automatically convert 'CRLF' (typical of Windows) line-endings into 'LF' (typical of Linux and macOS) when adding a file to the index (and vice versa when it checks out code onto your filesystem) → *git config --global core.autocrlf true* (in particular, when asserted on Windows machines, this converts 'LF' endings into 'CRLF' when you check out code);
- **NB #2**: 'CR' = '\r' = *Carriage Return* character | 'LF' = '\n' = *Line Feed* character.

◦ SSH key generation for encrypting and authenticating communication from/to server (assuming ED25519 algorithm):

- open Git bash (anywhere);
- type *ssh-keygen -t ed25519 -C "filippo.valmori@gmail"* specifying your email;
- empty-ENTER until completion;
- check public (.pub) and private keys have been successfully created in the specified (hidden) folder *.ssh* (e.g. *C:\Users\Filippo\.ssh*);

- from internet browser, go to *github website* > *profile icon* > *settings/manage-account* > *ssh keys* and upload the authentication public-key just created (simply drag-and-drop it);
- as a confirmation, type on bash *git gui* to open Git GUI and here go to *help* > *show ssh key* and verify the key has been successfully loaded.
- **NB #3**: setting up the SSH key is helpful because it allows to automatically authenticate yourself when accessing the remote server, thus without the need of supplying your username and password at each visit. For further details see *here*.

3. Locations & Syntax

○ Locally on each developer's PC the so-called *project directory* folder contains:

- ***working tree*** (WT), where the actual project files and directories (relative to a single commit at each time) are placed and can be edited;
- ***staging area*** (SA, sometimes called also *index*), where the file planned to be part of the next commit are stored (aka *staged*);
- ***local repository*** (LR), storing all the commits of the project (thus, representing its versioning history).

○ Note locally SA and LR are located inside the hidden sub-directory *.git*. Thus, removing this folder means removing all the project version history locally.

○ The ***remote repository*** (RR), unlike the other x3, is located remotely in a single data-center or cloud and represents the common interaction point among all developers (thus it identifies the official state of the project at any time). When LR and RR are synchronized, they contain exactly the same commits.

○ The general syntax for commands is *git [command] [--flags] [arguments]* or, more in detail, *git command (-f|--flag) [<id>] [<paths> ...]*, where:

- | = alternative;
- *[]* = optional values;
- - or -- = command flag or option;
- <> = required values (aka *placeholder*);
- *()* = grouping (for better clarity and disambiguation);
- ... = multiple occurrences possible.

References

[1] B. Sklar, P. K. Ray, *Digital Communications*, Chap. 4-9, Pearson Education, 2012.