6. Summary of Git terminology

6.1. Reference table with important Git terminology

The following table provides a summary of important *Git* terminology.

Table 1. Important Git terminology

Term	**Definition**
Branch	A *branch* is a named pointer to a commit. Selecting a branch in Git terminology is called *to checkout a branch*. If you are working in a certain branch, the creation of a new commit advances this pointer to the newly created commit. Each commit knows their parents (predecessors). Successors are retrieved by traversing the commit graph starting from branches or other refs, symbolic references (for example: HEAD) or explicit commit objects. This way a branch defines its own line of descendants in the overall version graph formed by all commits in the repository. You can create a new branch from an existing one and change the code independently from other branches. One of the branches is the default (typically named *master*). The default branch is the one for which a local branch is automatically created when cloning the repository.
Commit	When you commit your changes into a repository this creates a new *commit object* in the Git repository. This *commit object* uniquely identifies a new revision of the content of the repository. This revision can be retrieved later, for example, if you want to see the source code of an older version. Each commit object contains the author and the committer, thus making it possible to identify who did the change. The author and committer might be different people. The author did the change and the committer applied the change to the Git repository. This is common for contributions to open source projects.
HEAD	*HEAD* is a symbolic reference most often pointing to the currently checked out branch. Sometimes the *HEAD* points directly to a commit object, this is called *detached*

Term	**Definition**
	HEAD mode. In that state creation of a commit will not move any branch. If you switch branches, the *HEAD* pointer points to the branch pointer which in turn points to a commit. If you checkout a specific commit, the *HEAD* points to this commit directly.
Index	*Index* is an alternative term for the *staging area*.
Repository	A *repository* contains the history, the different versions over time and all different branches and tags. In Git each copy of the repository is a complete repository. If the repository is not a bare repository, it allows you to checkout revisions into your working tree and to capture changes by creating new commits. Bare repositories are only changed by transporting changes from other repositories. This book uses the term *repository* to talk about a non-bare repository. If it talks about a bare repository, this is explicitly mentioned.
Revision	Represents a version of the source code. Git implements revisions as *commit objects* (or short *commits*). These are identified by an SHA-1 hash.
Staging area	The *staging area* is the place to store changes in the working tree before the commit. The *staging area* contains a snapshot of the changes in the working tree (changed or new files) relevant to create the next commit and stores their mode (file type, executable bit).
Tag	A *tag* points to a commit which uniquely identifies a version of the Git repository. With a tag, you can have a named point to which you can always revert to. You can revert to any point in a Git repository, but tags make it easier. The benefit of tags is to mark the repository for a specific reason, e.g., with a release. Branches and tags are named pointers, the difference is that branches move when a new commit is created while tags always point to the same commit. Tags can have a timestamp and a message associated with them.
URL	A URL in Git determines the location of the repository. Git distinguishes between *fetchurl* for getting new data from other repositories and *pushurl* for pushing data to another repository.
Working	The *working tree* contains the set of working files for the repository. You can modify

Term	Definition
tree	the content and commit the changes as new commits to the repository.

6.2. File states in the working tree

A file in the working tree of a Git repository can have different states. These states are the following:

- untracked: the file is not tracked by the Git repository. This means that the file never staged nor committed.
- tracked: committed and not staged
- staged: staged to be included in the next commit
- dirty / modified: the file has changed but the change is not staged