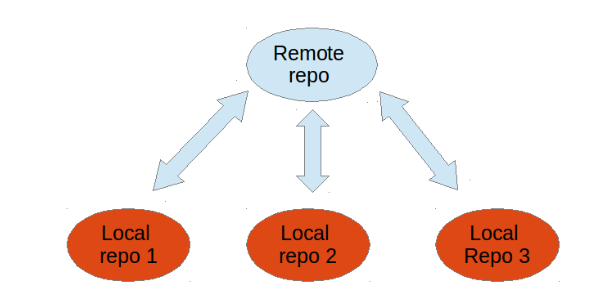
**[Distributed version control systems](https://www.vogella.com/tutorials/Git/article.html" \l "dvcs_definition)**

In a distributed version control system each user has a complete local copy of a repository on his individual computer. The user can copy an existing repository. This copying process is typically called cloning and the resulting repository can be referred to as a clone



Git is currently the most popular implementation of a distributed version control system.

### [Git repositories](https://www.vogella.com/tutorials/Git/article.html#gitdefintion_localrepositories)

A Git repository contains the history of a collection of files starting from a certain directory. The process of copying an existing Git repository via the Git tooling is called cloning. After cloning a repository the user has the complete repository with its history on his local machine. Of course, Git also supports the creation of new repositories.

### [Working tree](https://www.vogella.com/tutorials/Git/article.html#workingtree)

A local repository provides at least one collection of files which originate from a certain version of the repository. This collection of files is called the *working tree*. It corresponds to a checkout of one version of the repository with potential changes done by the user.

The user can change the files in the *working tree* by modifying existing files and by creating and removing files.

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

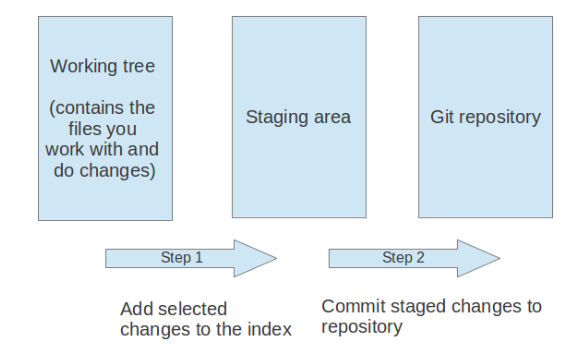
* untracked: the file is not tracked by the Git repository. This means that the file never staged nor committed.
* tracked: committed and not staged
* staged: staged to be included in the next commit
* dirty / modified: the file has changed but the change is not staged

### [Adding to a Git repository via staging and committing](https://www.vogella.com/tutorials/Git/article.html#gitaddingprocess)

After modifying your working tree you need to perform the following two steps to persist these changes in your local repository:

* add the selected changes to the staging area (also known as index) via the git add command
* commit the staged changes into the Git repository via the git commit command

This process is depicted in the following graphic.



### [Synchronizing with other Git repositories (remote repositories)](https://www.vogella.com/tutorials/Git/article.html#gitdefintion_remoterepositories)

Git allows the user to synchronize the local repository with other (remote) repositories.

Users with sufficient authorization can send new version in their local repository to to remote repositories via the *push* operation. They can also integrate changes from other repositories into their local repository via the *fetch* and *pull* operation.

### [The concept of branches](https://www.vogella.com/tutorials/Git/article.html#gitdefinition_branching)

Git supports *branching* which means that you can work on different versions of your collection of files. A branch allows the user to switch between these versions so that he can work on different changes independently from each other.

For example, if you want to develop a new feature, you can create a branch and make the changes in this branch. This does not affect the state of your files in other branches. For example, you can work independently on a branch called *production* for bugfixes and on another branch called feature\_123 for implementing a new feature.

Branches in Git are local to the repository. A branch created in a local repository does not need to have a counterpart in a remote repository. Local branches can be compared with other local branches and with *remote-tracking* branches. A remote-tracking branch proxies the state of a branch in another remote repository.

Git supports the combination of changes from different branches. The developer can use Git commands to combine the changes at a later point in time.

**Setup:**

1. **Installation**

A Windows version of Git can be found on the [Git download page](http://git-scm.com/downloads). This website provides native installers for each operating system. The homepage of the Windows Git project is [git for window](https://git-for-windows.github.io/).

1. **Configurations**

The git config command allows you to configure your Git settings. These settings can be system wide, user or repository specific.

Git allows you to store user settings in the .gitconfig file located in the user home directory. This is also called the *global* Git configuration. For example Git stores the committer and author of a change in each commit. This and additional information can be stored in the Git user settings.

### [User credential configuration](https://www.vogella.com/tutorials/Git/article.html#gitsetup_user):

You have to configure at least your user and email address to be able to commit to a Git repository because this information is stored in each commit.

**#** configure the user which will be used by Git

**#** this should be not an acronym but your full name

git config --global user.name "Firstname Lastname"

**#** configure the email address

git config --global user.email "your.email@example.org"

### [Ignoring files and directories with a .gitignore file](https://www.vogella.com/tutorials/Git/article.html#ignoring-files-and-directories-with-a-.gitignore-file)

### Git can be configured to ignore certain files and directories for repository operations. This is configured via one or several .gitignore files. Typically, this file is located at the root of your Git repository but it can also be located in sub-directories.

For example, the following .gitignore file tells Git to ignore the bin and target directories and all files ending with a ~.

**#** ignore all bin directories

**#** matches **"bin"** **in** any subfolder

bin/

**#** ignore all target directories

target/

**#** ignore all files ending with ~

\*~

To query your Git settings, execute the following command:

## [Exercise: Setting up Git via the command line](https://www.vogella.com/tutorials/Git/article.html#exercise_gitsetup)

After the installation of the Git tooling for the command line you need to configure Git.

The minimum required information to use Git is the user name and the email which Git should use.

You also configure Git to use rebase during a pull operation which is also a common setting for Git.

Configure your user and email for Git via the following command.

# configure the user which will be used by Git

# this should be not an acronym but your full name

git config --global user.name "Firstname Lastname"

# configure the email address

git config --global user.email "your.email@example.org"

# use rebase instead of merge in the `git pull` command.

# this avoids merge commits during the pull operation

git config --global branch.autosetuprebase always

## [Exercise: Performing a local Git workflow via the command line](https://www.vogella.com/tutorials/Git/article.html#firstgit)

In this exercise, you learn how to create and work with a local Git repository.

### [Create a directory](https://www.vogella.com/tutorials/Git/article.html#firstgit_directory)

The following commands create an empty directory which is used later in this exercise to contain the working tree and the Git repository.

**#** switch to the home directory

cd

**#** create a directory and switch into it

mkdir repo01

cd repo01

**#** create a new directory

mkdir datafiles

### [Create a new Git repository](https://www.vogella.com/tutorials/Git/article.html#firstgit_repository_creation)

You now create a new Git repository with a working tree.

Every Git repository is stored in the .git folder of the directory in which the Git repository has been created. This directory contains the complete history of the repository. The .git/config file contains the configuration for the repository.

Use the git init command to create a Git repository in the current directory. Git does not care whether you start with an empty directory or if it contains already files.

**# you should still be** in **the repo01 directory**

**cd ~/repo01**

**# initialize the Git repository**

**#** for **the current directory**

**git init**

### [Create new content](https://www.vogella.com/tutorials/Git/article.html#firstgit_content)

Use the following commands to create several new files.

**#** switch to your Git repository

cd ~/repo01

**#** create an empty file **in** a new directory (non-windows)

touch datafiles/data.txt

**#** create a few files with content

**ls >** test01

**echo "bar" >** test02

**echo "foo" >** test03

### [See the current status of your repository](https://www.vogella.com/tutorials/Git/article.html#firstgit_repostatus)

The git status command shows the status of the working tree, i.e. which files have changed, which are staged and which are not part of the staging area. It also shows which files have conflicts and gives an indication what the user can do with these changes, e.g., add them to the staging area or remove them, etc.

Run it via the following command.

git status

### [Add changes to the staging area](https://www.vogella.com/tutorials/Git/article.html#firstgit_repoadd)

Before committing changes to a Git repository, you need to mark the changes that should be committed with the git add command. This command allows adding changes in the file system to the staging area. It creates a snapshot of the affected files. You can add all changes to the staging area with the . option or changes in individual files but specifying a file pattern as option.

**#** add all files to the index of the Git repository

git add .

### [Commit staged changes to the repository](https://www.vogella.com/tutorials/Git/article.html#firstgit_repocommit)

After adding the files to the Git staging area, you can commit them to the Git repository with the git commit command. This creates a new commit object with the staged changes in the Git repository and the HEAD reference points to the new commit. The -m parameter (or its long version: --message) allows you to specify the commit message. If you leave this parameter out, your default editor is started and you can enter the message in the editor.

**#** commit your file to the **local** repository

git commit -m "Initial commit"

### [Viewing the Git commit history](https://www.vogella.com/tutorials/Git/article.html#viewing-the-git-commit-history)

The Git operations you performed have created a local Git repository in the .git folder and added all files to this repository via one commit. Run the git log command to see the history.

**#** show the Git log **for** the change

git log

### [Remove files](https://www.vogella.com/tutorials/Git/article.html#firstgit_deletefile)

If you delete a file, you use the git add . command to add the deletion of a file to the staging area.

## [Remote repositories](https://www.vogella.com/tutorials/Git/article.html#remotes)

Git supports several transport protocols to connect to other Git repositories; the native protocol for Git is also called git.

The following command clones an existing repository using the Git protocol. The Git protocol uses the port 9148 which might be blocked by firewalls.

**#** switch to a new directory

mkdir ~/online

cd ~/online

**#** clone online repository

git clone git://github.com/deepnitrkl/mytestrepo.git

Alternatively you could clone the same repository via the http protocol.

**#** the following will clone via HTTP

git clone http://github.com/deepnitrkl/mytestrepo.git

### [Adding remote repositories](https://www.vogella.com/tutorials/Git/article.html#remote_add)

If you clone a repository, Git implicitly creates a *remote* named *origin* by default. The *origin* *remote* links back to the cloned repository.

You can push changes to this repository via git push as Git uses origin as default. Of course, pushing to a remote repository requires write access to this repository.

You can add more *remotes* via the git remote add [name] [URL\_to\_Git\_repo] command. For example, if you cloned the repository from above via the Git protocol, you could add a new remote with the name *github\_http* for the http protocol via the following command.

**#** add the HTTPS protocol

git remote add github\_http https://xxxx@github.com/xxxxxxx/yyyyy.git

### [Rename remote repositories](https://www.vogella.com/tutorials/Git/article.html#remote_rename)

To rename an existing remote repository use the git remote rename command. This is demonstrated by the following listing.

**#** rename the existing remote repository from

**#** github\_http to github\_testing

git remote rename github\_http github\_testing

### [Using a proxy](https://www.vogella.com/tutorials/Git/article.html#remote_httpproxy)

Git also provides support for HTTP access via a proxy server. The following Git command could, for example, clone a repository via HTTP and a proxy. You can either set the proxy variable in general for all applications or set it only for Git.

The following listing configures the proxy via environment variables.

**#** Linux and Mac

export http\_proxy=http://proxy:8080

export https\_proxy=https://proxy:8443

**#** Windows

set http\_proxy http://proxy:8080

set https\_proxy http://proxy:8080

git clone http://git.eclipse.org/gitroot/platform/eclipse.platform.ui.git

The following listing configures the proxy via Git config settings.

**#** **set** proxy **for** git globally

git config --global http.proxy http://proxy:8080

**#** to check the proxy settings

git config --get http.proxy

**#** just **in case** you need to you can also revoke the proxy settings

git config --global --unset http.proxy

### [Synchronizing with remote repositories](https://www.vogella.com/tutorials/Git/article.html#remotes_remote_synchronize)

You can synchronize your local Git repository with remote repositories. These commands are covered in detail in later sections but the following command demonstrates how you can send changes to your remote repository.

**#** **do** some changes

**echo "I added a remote repo" >** test02

**#** commit

git commit -a -m "This is a test for the new remote origin"

**#** to push use the **command**:

**#** git push **[**target]

**#** default **for** **[**target] is origin

git push origin

### [Show the existing remotes](https://www.vogella.com/tutorials/Git/article.html#remotes_showremote)

To see the existing definitions of the remote repositories, use the following command.

**#** show the details of the remote repo called origin

git remote show origin

To see the details of the *remotes*, e.g., the URL use the following command.

**#** show the existing defined remotes

git remote

**#** show details about the remotes

git remote -v

### [Push changes to another repository](https://www.vogella.com/tutorials/Git/article.html#cloneremotes_push)

The git push command allows you to send data to other repositories. By default it sends data from your current branch to the same branch of the remote repository.

### [Pull changes from a remote repository](https://www.vogella.com/tutorials/Git/article.html#cloneremotes_pull)

The git pull command allows you to get the latest changes from another repository for the current branch.

## [Using Branches](https://www.vogella.com/tutorials/Git/article.html#using-branches)

Git allows you to create branches, i.e. named pointers to commits. You can work on different branches independently from each other. The default branch is most often called master.

### [List available branches](https://www.vogella.com/tutorials/Git/article.html#gitbranch_listbranches)

The git branch command lists all local branches. The currently active branch is marked with \*.

**#** lists available branches

git branch

If you want to see all branches (including remote-tracking branches), use the -a for the git branch command. See [Remote tracking branches](https://www.vogella.com/tutorials/Git/article.html#gitremotebranch_overview) for information about remote-tracking branches.

**#** lists all branches including the remote branches

git branch -a

### [Create new branch](https://www.vogella.com/tutorials/Git/article.html#gitbranch_createnewbranch)

You can create a new branch via the git branch [newname] command. This command allows to specify the commit (commit id, tag, remote or local branch) to which the branch pointer original points. If not specified, the commit to which the HEAD reference points is used to create the new branch.

**#** syntax: git branch <name> <**hash>**

**#** <**hash>** **in** the above is optional

git branch testing

### [Checkout branch](https://www.vogella.com/tutorials/Git/article.html#gitbranch_checkout)

To start working in a branch you have to *checkout* the branch. If you *checkout* a branch, the HEAD pointer moves to the last commit in this branch and the files in the working tree are set to the state of this commit.

The following commands demonstrate how you switch to the branch called *testing*, perform some changes in this branch and switch back to the branch called *master*.

**#** switch to your new branch

git checkout testing

**#** **do** some changes

**echo "Cool new feature in this branch" >** test01

git commit -a -m "new feature"

**#** switch to the master branch

git checkout master

**#** check that the content of

**#** the test01 file is the old one

cat test01

To create a branch and to switch to it at the same time you can use the git checkout command with the -b parameter.

**#** create branch and switch to it

git checkout -b bugreport12

**#** creates a new branch based on the master branch

**#** without the last commit

git checkout -b mybranch master~1

### [Delete a branch](https://www.vogella.com/tutorials/Git/article.html#gitdeletebrach)

To delete a branch which is not needed anymore, you can use the following command. You may get an error message that there are uncommited changes if you did the previous examples step by step. Use force delete (uppercase -D) to delete it anyway.

**#** delete branch testing

git branch -d testing

**#** force delete testing

git branch -D testing

**#** check **if** branch has been deleted

git branch

### [Push changes of a branch to a remote repository](https://www.vogella.com/tutorials/Git/article.html#gitpushbranch)

You can push the changes in a branch to a remote repository by specifying the target branch. This creates the target branch in the remote repository if it does not yet exist.

If you do not specify the remote repository, the origin is used as default

**#** push current branch to a branch called **"testing"** to remote repository

git push origin testing

**#** switch to the testing branch

git checkout testing

**#** some changes

**echo "News for you" >** test01

git commit -a -m "new feature in branch"

**#** push current HEAD to origin

git push

**#** make new branch

git branch anewbranch

**#** some changes

**echo "More news for you" >>** test01

git commit -a -m "a new commit in a feature branch"

**#** push anewbranch to the master **in** the origin

git push origin anewbranch:master

**#** get the changes into your **local** master

git checkout master

git pull

This way you can decide which branches you want to push to other repositories and which should be local branches