```
/*****************************************************
*                        Git and Github – Command
*****************************************************/

$ git –version

$ git init                                -- Creates a repository and a .git folder. Doesn't perform the initial commit.

$ git log                                 -- Shows log on currently checked out branch.

$ git log <branch>|<remote>/<branch>      -- Shows log on <branch>.

$ git log -n 1                            -- Show n commits.

$ git log --stat                          -- Show filename in each commit + number of changes in each file.

$ git log --name-only                     -- Show filename in each commit.

$ git log --graph <branch_1> <branch_2>   -- Show branches visually. Try adding a --oneline to make it easier to read.
                                          -- You can graph more than two branches (if you like).

$ git diff <older_id> <newer_id>          -- Difference between commits <older_id> and <newer_id>.
$ git diff                                -- Difference between the Working directory and Staging area.
$ git diff --staged                       -- Difference between the Staging area and Repository.
$ git show                                -- Diff last commit with its parent
                                          -- (Please note: PARENT!!!, not necessarily the commit prior to the one you are interested in -
                                          -- think merging!).

$ git show <commit_id>                    -- Diff between <commit_id> and its parent (Please note: PARENT!!!, see above for details).

$ git reset --hard                        -- Revert changes in Working directory and Staging area. Irreversible change!!!
$ git reset <file>                        -- Removed file from staging. Changes are kept.
$ git clean -f                            -- Remove EVERYTHING, including untracked files (e.g., new files, generated files)
$ git clean -f -X                         -- Remove EVERYTHING, including ignored and untracked files (e.g., new files, generated files)
$ git revert -n <commit_id>               -- Revert, but keep in Working Copy (do not commit reverted version automatically).
$ git revert -m 1 <commit_id>             -- Revert a merge commit. Reverts all commits that were part of that merge.
                                          -- You can't revert a merge commit if you used fast-forward commit.

$ git branch                              -- View branches on the current repository.
$ git branch <name>                       -- Create new branch (this branch will not be checked out automatically) on the current
                                          -- repository, from the current HEAD. It essentially labels the current head with <name>.
$ git branch -d <branch_name>             -- Create a branch from detached HEAD. It is the same thing as doing the following set of
                                           -- commands:
                                          --    $ git branch <new_branch_name>
                                          --    $ git checkout <new_branch_name>

$ git merge <branch_1> <branch_2> ...     -- Merge specified branches into currently checked out branch.
$ git merge --abort                       -- Revert branches to state before the merge. Useful if you have a merge conflict.

$ git checkout -b <name> <from_branch>    -- Create new branch and check out automatically.
$ git checkout <commit_id>                -- Use an older commit (detached HEAD state).
$ git checkout master                     -- Use last commit as HEAD.
$ git checkout -b <name>                  -- Use with detached HEAD state, in a situation where you added new commits to the
                                             detached HEAD and now want to make it into a new branch.

$ git stash                               -- Git moves uncommited changes along when you switch branches. If you want to "save" the
                                          -- changes without committing them,

$ git stash pop                           -- you have to stash them. Once you're done working on the other branch, you can retrieve the
                                          -- changes. If you have created new files (but haven't committed them yet), you must first stage
                                          -- them before you can stash them.

$ git rc                                  -- Garbage collection (removing deleted branches whose commits have not been merged and
                                          -- are therefore unreachable).
```

Committing process.

```
$ git status                        -- 1. Shows current branch working directory and staging area,
                                    --    changed files, latest commit, untracked files. Also shows if there is any difference
                                    --    in number/status of commits between local repo and repo on GitHub.
$ git add <filename>                -- 2. Add file to Staging area.
$ git diff                          -- 3. Difference between the Working directory and Staging area.
$ git commit                        -- 4. Commits to repository. If a branch is checked out, it will commit to that branch.
```

Merging process

```
$ git checkout <branch>             -- 1. checkout <branch> you want to merge into.
$ git merge --no-ff <branch_1>      -- 2. merge the branche into the checked out branch.
$                                   -- 3. resolve conflicts by opening the conflicted file. 3 sections:
                                    --    <<<<<<< HEAD     <branch we're merging into>
                                    --    |||||||                      merged common ancestor
                                    --    =======
                                    --    >>>>>>> master   <branch we're merging from>
$ git add <files>                   -- 4. add files to Staging area.
                                    --    Conflict resolution is also signalled this way (no special "Resolved" option).
$ git rebase master                 -- Alternatively, you can rebase current branch on the tip of master.
```

Creating a repository on GitHub and connecting it with our local repo - first approach.

```
$                                         -- 1. Create a new repo on GitHub directly (via GitHub website).
                                          --    Give it any name, e.g. "reflections".
$ git init                                -- 2. Create a local directory and run this command in it.
$ git remote add <remote_repo_name> <url> -- 3. Add the remote repository (found in <url>) to the local repository
                                          --    and name it <remote_repo_name>. <remote_repo_name> is a way to reference
                                          --    the remote repo from within current local repository.
                                          --    <remote_repo_name> is usually "origin" if we have only one remote.
                                          --    Remote repo is a version/representation of the local project (repo),
                                          --    but stored on a server. When the branch is pushed, the remote repo is
                                          --    named same as the branch.
                                          --    For simplicity, use HTTPS!
$ git remote                              -- 4. View all remotes (created by you or by repository owner).
$ git remote -v                           -- 5. Check if URL was added correctly.
                                          --    Shows the URL you will fetch from and the URL you will push to.
```

Creating a repository on GitHub and connecting it with our local repo - second approach

```
$                                   -- 1. Create a new repo on GitHub directly (via GitHub website).
                                    --    Give it any name, e.g. "reflections".
$ git clone <url>                   -- 2. Downloads a repository. It also sets up the remote to point to <url>.
```

Communicating with the repository

```
$ git push <target_remote> <branch_to_push>  -- Push branch to remote. Branch on remote repo will have the same
                                             --    name as the local branch that was just pushed.
$ git pull <target_remote> <remote_branch>   -- Pull commits from remote repo's <branch> to a local <branch> of the same name.
                                             --    e.g. $ git pull origin master -> local master
                                             --    Merge differences immediately.
                                             --    If local branch HEAD is an ancestor of new commits, then a "fast-forward
                                             --    commit" is done.
                                             --    If local branch HEAD is not an ancestor of new commits (local and
                                             --    remote branch have diverged,
                                             --    NO conflicting changes introduced), a new merge commit is created.
                                             --    Same as: (master)$ git fetch origin + git merge master origin/master
```

If local and remote repo have diverged and you are NOT aware of it!

```
$                                            -- See above about pulling. You will have to resolve and then proceed with staging
                                             -- and commiting.
                                             -- No big deal. Below is a more in-depth approach (same outcome).
```

If local and remote repo have diverged and you are aware of it!

```
(master)$ git fetch origin                -- Pull commits from remote repo's branch with the same name as the local checked
                                          -- out branch:
                                          --    e.g. GitHub origin/master -> Local origin/master
                                          --    Updates local version of origin/<branch>. Does not affect your local <branch>,
                                          --    only the local origin/<branch>
                                          --    You can check log for your local origin/<branch> by doing $ git log
                                          --    origin/<branch>
                                          --    Does not merge local <branch> and origin/<branch>!
$ git merge master origin/master          -- Merge. Might warn of a conflict.
$ code <conflicted_file>                  -- Edit conflicted file.
$ git add <conflicted_file>               -- Signal conflict resolution by staging the file.
$ git commit                                        -- Commit the resolved file.
```

Forking a repository on GitHub.

```
$                                         -- 1. Go to GitHub and press "Fork" (upper left corner).
$ git clone <url>                         -- 2. Download repo to local computer.
                                          --    Remote repo is already added, pointing to original repo on GitHub.
$                                         -- 3. Add collaborators: GitHub repo -> Settings -> Collaborators.
```

Pull request

```
$                                         -- A request towards someone (branch owner) to review and merge our branch.
                                          -- It can also be thought of as a "merge request".
                                          -- Every step is done on GitHub:
                                          -- 1. Choose a branch you want the Pull request to be created for.
                                          -- 2. Choose "Pull Request" option.
                                          -- 3. GitHub assumes you want the original repository (if you forked)
                                          --    to be the destination repo.

                                          --    Set base fork to be e.g. master.
                                          -- If the branch you are requesting to merge into has had additional commits
                                          -- that will cause a conflict, you will have to resolve this locally. Please consult
                                          -- the "Pull requests and conflicts" section further below.
```

```
/*******************************************************
*                       Git and Github – Concepts
*******************************************************/

# Row width: 80.          -- Try to keep row width to 80 chars, it helps Git visualize changes better.

# Working directory       --

# Staging Area            -- Contains a copy of your local repository. When you stage a certain change, it gets moved to this staged
                          -- copy. When you commit the staged change, it remains in the Staging area.
                          -- Commit makes the staged area and local repository equal.

# Local repository    --

# Remote repository  --

# Reachability            -- Each commit has a parent. Each commit stores its commit parent.
                          -- When you commit, current head becomes the new commit's parent (head moves, of course).
                          -- Log shows commits starting with the head and goes back to the first commit that does not
                          -- have a parent (usually this is the initial commit). Commits in different branches are
                          -- not visible from one another - this is what "reachability" means. When you do a commit
                          -- from the detached head state and then checkout an existing branch, that commit is now
                          -- lost, since it is not reachable from any of the current branches (to better visualize
                          -- this: create a commit graph with two concurrent branches and a commit from some detached
                          -- head. Now checkout one of the branches - there is no way for you to see that lost commit in
                          -- logs and you cannot do a checkout using branch names - you can do a checkout if you
                          -- remember the commit id).

# HEAD                    -- Current commit. When you make a new commit, head is moved to this new commit.
```

# Detached HEAD	-- this means that we're looking at a commit that was not labeled with a branch name. -- We can create a branch from this using: git branch -d <branch_name>
# Branch name	-- A branch is actually a labeled commit. Head commit of a particular branch is the one -- that is labeled. If the head is the same commit that is labeled as branch, when you -- commit, the label moves to the new head. -- Commits themselved do not know anything about the branches they belong to.
# Head and checkout	-- When you do a checkout, you make some commit the new Head. Which commit? This depends: if you are -- checking out a branch, then the new head becomes the branch's head. If you do a checkout on a specific -- commit id, then that commit becomes the new head (detached HEAD state).
# Merge branch branch.	-- Once one branch is merged into another, all the commits from the merged branch are visible in the main -- Merge process compares three commits: heads of both branches and their common parent. -- Merge and reachability: merge commit has two parents (one from each branch).
# Cloning	-- A Git concept. We can clone a remote repository (from GitHub url) to our local computer. -- We can clone a repo from a local computer as another (new) local repo.
# Remote branch	-- A branch created on the remote repository. We can do a checkout and use it as if it were a regular -- branch. -- Git stores locally state of all remote branches:<remote>/<branch>. Local Git stores last known position -- (commit ID) and the repository. This way when you do a "git fetch", you get all the newest changes from -- origin <branch>, but this does not affect your local <branch>, only local origin/<branch>. -- State is updated everytime we push or pull.
# Fast-forward merge	-- Occur when one commit (the one with the branch tag) is the ancestor of another commit (the other -- branch tag). When merging, if local branch HEAD is an ancestor of new commits, then a "fast-forward -- commit" is done (no new commit). It simply moves the HEAD of the current branch. -- It is easy to do a fast-forward merge when you first perform a rebase. -- HOWEVER, if you merge on GitHub, it does create a new commit (even if the new commit wasn't necessary).
# Pull request have sufficed.	-- A purely GitHub concept! Merging a pull request results in a new commit, even if a ff-commit would
# Pull requests and	-- All such conflicts must be resolved LOCALLY. GitHub will notify you of conflicts, but it will not -- conflicts resolve conflicts - you must pull conflicting branches and resolve locally. PLEASE CONSULT -- POINTS BELLOW.
# Pull request conflict in your repo	-- Merging directly on GitHub is not allowed in such cases. Such conflict must be resolved locally. -- You must first merge master into branch LOCALLY. Then push (this updates the pull request). -- Only then can you merge the pull request directly on GitHub.
# Pull request conflict	-- So you have a fork and on it a branch - and want to do a pull request from the branch towards -- fork's original repo, but there are conflicting changes present. You first create a new remote to point -- repo to original repo, called "upstream". Checkout the master. Do a "$ git pull upstream/master" to -- update the local master branch to the latest commit on the original repo. Merge master into branch. -- Push branch (this updates the pull request as well). Your branch is now up-to-date with the -- original repo's + it has your changes that you want to pull into original repo.
# Rebase	-- Useful for integrating smaller feature branches. For longer-running feature branches, use 3-way --no-ff -- merge commits.
# Forking	-- A purely GitHub concept! Cloning a repository directly on GitHub, under your own account. -- When you do a fork, it is customary to immediately create another branch. This way -- your master can be kept synchronized with the original repo and branch can be used for development. -- You fork directly via GitHub web UI.
# Collaborators	-- A purely GitHub concept! A list of people you allow psuhing to your repository. -- Settings -> Collaboration