Mechatronic Engineering

Object Oriented Programming and Software Engineering
Laboratory instruction 4
Git - distributed version control system - Part 1

AGH Kraków, 2021

Table of contents

1 Git - a distributed version - control system

Git is a tool for tracking changes made over time to a set of files. This is the so-called version control system. It is most often used to coordinate the teamwork of a group of programmers. Git does not have a central server where the project is saved. Users work on their own copy of the repository independently of each other, which is why it is a distributed system. For more detailed information use git manual by typing in the terminal *man git* or *git --help* for help.

2 Getting started

Before one starts working with git, several configuration parameters must be set. Command *git config* is used to load and manipulate git settings (local for single repository, global for all user repositories, and system level). Git settings are stored in *~/.gitconfig*. It is a standard text file that can be edited by any text editor. One changes parameter values by typing the following command in the terminal:

```
$ git config --{local,global,system} parameter value
```

When using Git it is important to identify the person who made changes to the repository, especially when one works on a big project that involves many people. That is why first parameters to change should be the user name and email address. One can do it by typing into terminal:

```
$ git config --global user.name "username"
$ git config --global user.email email@address
```

Run above commands to set your personal data into your *git config* file.

3 Creating new repository

To create a new repository, one needs to type the command:

```
$ git init folder_name
```

if there was no folder named *folder name* it will be created with *.git* folder inside. Git stores the new empty repository in *.git* folder. In case in which *folder name* does not exist, it will be created.

4 Adding files to the repository

To add files stored in *folder name* to your repository one needs to use command:

```
$ git add file_name
```

Where *file name* is the name of the file one want to add to the repository. To add all files stored in *folder name* to your repository one needs to type in the terminal:

```
$ git add .
```

5 File exclusion

There are file types that don't require to be added to the repository. Those files are binaries, compilation artefacts, all automatically generated configuration tools, etc. Examples are listed bellow:

object code: *.o, *.so, *.a, *.dll, *.exe byte
code: *.jar, *.elc, *.pyc
compilation system artefacts: config.log, config.status, aclocal.m4, Makefile.in

Generally, files that don't suppose to be a part of a repository are files that don't require revisions or version control. There are three methods to exclude files from repositories.

- Firstly, one can list excluded file names in file *.gitignore* in the work tree. From git perspective, *.gitignore* is a standard text file, so it is possible to add it to the repository. *.gitignore* can also be listed in *.gitignore*.
- Secondly, one can list excluded files in file *.git/info/exclude*. The difference from previous method is that it can't be synced via repository.

- Thirdly one can list excluded file names in configuration variable *core.excludes* by command:

```
    $ git config --global core.excludefile ~/.gitignore
```

above example bases on the assumption, that *home* directory is not a part of repository.

Form more detailed description and file structure type in the command line `man gitignore`.

6 Removing files

To remove file from the project command *git rm* should be used:

```
$ git rm file_name
```

This command makes two operations:
Removes entry from the index (this change applied in next approval).
Deletes file from the work tree (with the use of *rm file name*) as in Linux Console

7 Changing file name

To change file name in the project command *git mv* should be used:

```
$ git mv original_file_name new_file_name
```

Above command works as set of commands: To remove file from the project command *git rm* should be used:

```
$ mv original_file_name new_file_name
$ git add new_file_name
```

8 Withdrawal of modifications from the index

There is a possibility to withdraw changes made to repository by the use of:

```
$ git reset
```

It will reset all changes in the project to last approved state. Additionally, git will display the list of changes that will be withdrawn.

9 Displaying changes log

To display the full log of changes made to the project one needs to use:

```
$ git log --stat
```

10 Approving changes

This section provides information on how to approve changes made to your project into the repository. Command *git commit* is used for approving changes to the repository. When one types the command without any additional arguments all changes are accepted. To approve only specific files, one needs to use:

```
$ git commit -{flag} {option} file_name
```

Comand *git commit* creates a new object of repository tree, capturing the current state of the index and change approval object with the text of the comment and the author's data, date, etc.

```
$ git commit -m 'your_message'
```

If flag *-m* will be omitted, no message will be attached. Command *git commit* has many helpful flags, to display detailed description of its type *git commit --help* into terminal. Before one approves changes, it is helpful to verify the changes made to the project by using:

```
$ git status
```

and

```
$ git diff
```

The presented commands are used to display the awaiting changes status (*git status*) and differences between modified and lastly accepted file (*git diff*).

11 Cloning repositories

To use an existing repository on one's computer it has to be cloned. Command *git clone* is responsible for creation of new clone. The original repository will be called *remote* repository. The URL provided to the command can origin from multiple online protocols like HTTP, FTP and SSH.

```
$ git clone remote_repository_adres
```

Method for cloning repositories through SSH will be presented for the purpose of this instruction. It will simplify the process of uploading your code to the AGH SSH account from your home computer.
To clone repository from AGH SSH one needs to input *git clone* with a proper formatting:

```
$ git clone ssh://user@student.agh.edu.pl:/your/repository/location
```

To obtain proper path to the repository one can type *pwd* while the prompt is in the repository directory.
WARNING! The path to the repository must be specified in the full form (egz. /home/imirgrp/user/localRepoDir.git). In case in wich repository is not recognised try the path without *.git*.
To update your clone, one needs to use *git pull* command:

```
$ git pull
```

12 Pushing repositories to a remote location

Pushing repositories is required when one wants to upload approved changes in their local project to a remote repository. To do it *git push* command has to be used.

```
$ git push
```

The above command will push all the approved changes into origin location of the repository (if the repository was a clone the push will be performed to the remote location). One can also push the repository to a custom addressed repository. It can be used to create a new remote repository for our project to be cloned by other project users:

```
$ git push repo_url
```

12.1 Pushing to AGH SSH workaround

In some cases, pushing your local copy to the remote repository on AGH servers generates an error:

```
$ git push
user@student.agh.edu.pl's password:
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Writing objects: 100% (3/3), 227 bytes | 227.00 KiB/s,
done. Total 3 (delta 0), reused 0 (delta 0) remote:
error: refusing to update checked out branch:
    refs/heads/master
remote: error: By default, updating the current branch in a
    non-bare repository
remote: is denied, because it will make the index and work tree
    inconsistent
remote: with what you pushed, and will require 'git reset --
    hard' to match
remote: the work tree to HEAD.
remote:
remote: You can set the 'receive.denyCurrentBranch'
    configuration variable
remote: to 'ignore' or 'warn' in the remote repository to allow
    pushing into
remote: its current branch; however, this is not recommended
    unless you
remote: arranged to update its work tree to match what you
    pushed in some
remote: other way.
remote:
remote: To squelch this message and still keep the default
    behaviour, set
remote: 'receive.denyCurrentBranch' configuration variable to
    'refuse'.
To ssh://user@student.agh.edu.pl:/your/repository/location
 ! [remote rejected] master -> master (branch is currently
     checked out)
error: failed to push some refs to
    'ssh://user@student.agh.edu.pl:/your/repository/location'
```

There is a workaround to resolve this issue. Git operates on branches, while not going into details, the error forbids the user from pushing to master branch.

The workaround is to create a new branch (on your local machine) for the project and work on your local machine with usage of this branch:

```
git checkout -b localRepo
```

The git checkout command lets you navigate between the branches created by git branch. Checking out a branch updates the files in the working directory to match the version stored in that branch, and it tells Git to record all new commits on that branch [1].

After creating a new branch one can push the local repository to the remote. Now our local files are stored on remote repository, but the changes are not made to the master branch, so the next step will be merging our *localRepo* branch with *master*. Before you do that please check the output of following command:

```
$ git log --graph --all
```

The result will display the graph of our changes to the repository. The merging must be performed on the server machine. To merge *localRepo* branch with master one needs to type the following:

```
$ git merge localRepo
```

WARNING! Be aware that the presented method is shown due to the need for remote work in the current crisis and should not be considered as acting in accordance with the art. Also, if changes are made to the files on remote repository (that checkouts master) it can generate trouble. In this workaround it is preferred to work with the code only on your personal computers and use the server to merge the branch with master. It is possible to mix work (local/server) but it requires additional knowledge on working with branches. You do it at your own risk.

Task

For the purpose of this laboratory create a git repository of your software prepared for previous classes and upload it to a remote repository on your SSH account. The file structure of your repo should contain all the laboratory tasks in separate folders (lab01, lab02 etc.).

Bibliography

[1] https://www.atlassian.com/git/tutorials/using-branches/git-checkout
[2] R. Silverman, Git Pocket Guide, helion, 2014.
[3] W. Gajda, Git Rozproszony system kontroli wersji, helion, 2014.
[4] https://www.youtube.com/watch?v=USjZcfj8yxE
[5]https://www.notion.so/Introduction-to-Git-ac396a0697704709a12b6a0e545db049
[6] https://www.youtube.com/watch?v=HVsySz-h9r4