


SOFTWARE ENGINEERING

REVISION CONTROL SYSTEM LABORATORY

GIT system - fundamentals

version 0.4e

prepared by:
Radosław Adamus

This work is licensed under a  [CC BY 3.0](https://creativecommons.org/licenses/by/3.0/).

HISTORIA WERSJI

DATA	WERSJA	AUTOR	OPIS
08.11.2013	0.1	Radosław Adamus	Pierwsza robocza wersja dokumentu
17.11.2013	0.2	Radosław Adamus	Uzupełnienie opisu, dodanie zadania "Praca zespołowa", poprawki edycyjne, uzupełnienie informacji licencyjnych
23.11.2013	0.3	Radosław Adamus	Modyfikacje na podstawie doświadczeń pierwszego laboratorium: częstsze zatwierdzanie zmian, zwrócenie uwagi na kodowanie pliku, poprawki w linkach, poprawki edycyjne, zmiany nazw plików opisujących zmiany w przykładowym projekcie
27.03.2014	0.4e	Tomasz Kowalski	Translation into english

Goal:

The laboratory goal is to:

1. familiarize with basic functionalities of GIT revision control system and introduction to GitHub platform,
2. understand the relevance of version control and the necessity of revision control system in individual and team work.

Effects:

After laboratory student can:

1. initialize remote and local repositories, combine them together and send information between them,
2. add files to a repository and confirm changes,
3. create branches, switch between them and merge introduced changes.
4. take advantage of GitHub platform mechanisms to create project releases and realise basic teamwork scenario.

Prerequisites:

Registered a GitHub account (<https://github.com/>).

Tools:

Local tasks are conducted using system command-line [git-scm](#) and graphic tool git-gui (e.g. [TortoiseGit](#)¹). It is assumed that operations are performed from the command line, and the graphical interface tool is used to visualize the graph version (called revision graph).

Command required during classes:

- [git add](#)
- [git checkout](#)
- [git clone](#)
- [git commit](#)
- [git config](#)
- [git branch](#)
- [git init](#)
- [git merge](#)
- [git push](#)
- [git rebase](#)
- [git remote](#)
- [git status](#)

¹or other (TortoiseGit is Windows application) which can visualize revision graph (https://git.wiki.kernel.org/index.php/InterfacesFrontendsAndTools#Graphical_Interfaces).

Rules of laboratory exercise:

1. Commenting changes while committing should be meaningful. Introduced information should explain what a change concerns.
2. Repository should be configured so changes are signed by a user name (corresponding to name, surname) and e-mail address.
3. Final result must be located on a student account in GitHub system. When not all tasks are finished during classes changes should be committed and current state of local repository sent to remote repository.
4. History of changes (together with comments) will be controlled - it must correspond to a structure of tasks.
5. OPTIONAL: During classes display revision graph (TortoiseGit -> Revision graph) and record its state for particular tasks as graphic files (File -> Save graph as). It will facilitate understanding of configuration management rules.

Introduction:

1. What is a revision control

Revision control is an aspect of (software) configuration management associated with change management within electronic documents.

2. What is a revision control system

Revision control system is a software tool facilitating revision control process. The most important of its features are store (repository) management and access control. Access to the resource can mean downloading its specific version, the introduction of a new version directly or through a combination of changes from other versions of the resource.

3. What is a distributed revision control system

In a distributed revision control system many parallel repositories with different versions of documents can exist. Besides committing changes such systems enable synchronization between repositories.

Description of the laboratory:

0. Introductory remarks

The structure of laboratory separates the problem of revision control system (which is the main topic) from a project being controlled (which is only an example - context). Changes, which are to be introduced within the example project, are described independently in individual files.

During the exercise, each student uses a remote repository (public) stored on GitHub platform, and a local repository (private) located on the workstation. Most of the activities are related to version management at the local level. Synchronizing local and remote repository is done only at certain times specified in the instructions (and/or at the end of classes).

I First part - scenario for individual work

1. Remote initialization

Create a project repository (*repo in short*) in GitHub system named iislabpio_*[index]*.

2. Local initialization of project repository

Create a project folder and initialize git repo inside (git init).

Locally configure your user name (giving yours name and surname) and e-mail address (git config).

Create a file named 'README.md' and add it to the local repo (git add). Edit file entering your index number and laboratory group. Confirm changes (git commit) remembering about appropriate comment. Merge local repo (default branch *master*) with remote (git remote). Send (git push) the local branch master to remote repo.

The structure of a project and work branches

Locally create a beginning version of the project as described in the file *init.txt*. Test the correctness of the changes. Commit the changes (git commit) mindful of the comments.

Create branches 'feature1' and 'feature2' (git branch).

Switch to the *feature1* branch (git checkout) and introduce **two first** changes as described in the file *changes_feature1.txt*. Make changes in a sequence, according to the points of description, after each step testing and committing them in a repository.

Switch to the *feature2* branch and introduce **two first** changes as described in the file *changes_feature2.txt*, testing them and committing in a sequence (similarly as for *feature1*).

Merging changes in a master branch

During testing you should notice that the basic structure of the project contains a mistake. Corresponding correction should be included in the master branch of the project before work on individual features will be completed. Therefore, you should first and foremost make changes to the master branch. To avoid the lack of cohesion between branches you should also merge changes from master branch to the feature branches.

Switch to the master branch and make changes as described in the file *bugFix_master.txt*. Test the correctness of the changes. Commit the changes.

Switch to the *feature1* branch and merge changes from master branch using **git rebase** command. Observe the messages displayed during the operation. With the use of the tools available display a revision graph.

Now perform merging for the *feature2* branch, but this time use **git merge** command. Observe the messages displayed during the operation. [What are the differences between merge and rebase command?](#) With the use of the tools available view a revision graph version.

Finalize work in branches *feature1* and *feature2*

Complete implementation of *feature1* and *feature2* branches according to the remaining steps in files *changes_feature1.txt* and *changes_feature2.txt*.

Merging changes to master branch

Work on features has been completed and the contents of the main branch of the project (master) should be merged with the contents of branches *feature1* and *feature2*. Complete the merge (using one of the methods: rebase or merge).

Test the correctness of the merged structure. With the use of the tools available display a revision graph. Will the graph be the same regardless of the merge method chosen?

Finalizing and publishing changes

Enter (in the master branch) changes as described in the file *changes_master_part1.txt*. Test the correctness of the changes. Commit the changes.

Send the result to the remote repository.

In the final step, make the change as described in the file *changes_master_part2.txt*. Test the correctness of the changes. Commit the changes.

Again, send a revised version to the remote repository.

Using GitHub system create the release marking it as v0.1

II Second part - teamwork scenario

[The exercise requires performing the tasks of the first laboratory part]

Create a project on yours GitHub account based on a project (GitHub command: fork -

<https://help.github.com/articles/fork-a-repo>) of colleague on your right (if you are the last person in the row - the person sitting to your right is the first person in the second row). If a person has not yet completed the previous tasks - first her help. Then, make a clone of created remote repository to your local computer. Enter changes in the project as described in the file *changes_fork.txt*. Test the correctness of the changes. Commit the changes. Send the result to the remote repository. Next, send to the colleague information about changes you made ([pull request](#)).

[Merge changes made by a pull request](#) to your repository. Create a new release and mark it as v0.2.

Supplementary questions

1. Which git command is used by a GitHub platform to create a release?

2. [How to roll back changes in the file in the working copy and not stored in the repository \(or restore the latest version from the repository\)?](#)

3. [How to roll back the changes already recorded in the repository so the latest version of branches corresponds to changes introduced N commits before? Explain the two methods, the differences between them and, due to the differences, the range of applicability.](#)

4. How do I go back to the previous version in the repository? How do I go back N versions?

Interactive courses:

<http://try.github.com/>

<http://pcottle.github.io/learnGitBranching/>

Materials:

<http://git-scm.com/book>

<http://githubtraining.s3.amazonaws.com/github-git-training-slides.pdf>

<http://ndpsoftware.com/git-cheatsheet.html>

<https://commons.lbl.gov/display/pbddocs/Learn+Git>