Git

## Prerequisites:

Into:

<https://guides.github.com/introduction/git-handbook/>

Git Quick Learn Course

<http://gitimmersion.com/> (Press: Start the course)

<https://git-scm.com/book/en/v2> (Pro Git)

Supplementary references:

Interactive tutorials:  
 <https://learngitbranching.js.org/?demo>

<https://learngitbranching.js.org/>

<https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud>

Books & Guides:

<http://rogerdudler.github.io/git-guide/>

<https://examples.javacodegeeks.com/enterprise-java/maven/maven-settings-xml-example/>

## Lecture 3

### Lecture is divided into 3 parts:

* Overview, theory, demonstration
* Intellij Idea GIT support
* Practical part with GitHub (knowledge polishing)

### Git basics:

1. Git is an example of a distributed version control system

SVN is an example of centralized VCS

VCS allow:

* See commits
* Work and collaborate from different time zones

1. Git repository = Git Project

Git is a self-contained project.

Git is good for terminal usage. GIT GUI for history review.

Git operates and Git building blocks:

* Commits
* Branches
* Operations (clone, init, add, commit, status, branch, checkout, pull, fetch, push, merge, rebase, cherry-pick, log and etc.)

1. Git allows to keep a snapshot of the project without history by forking the project.
2. Secure access by SSH keys (local private and aligned public)

# Introduction

Git consists of 3 repos:

* Local (file directory - untracked)
* Index (files are put into local repo and hashed, tracked, commits can be reset)
* Git repo (pushed for)

### Commands:

1. $ git config --global user.name (check the user settings)
2. $ git config --global user.name "John Doe"
3. $ git config --global user.email johndoe@example.com
4. init - prepare for tracking a project (directory) for a version control system
5. clone - copy existing project under VCS into local directory (history, file and branches will be visualized, also set up default branches for tracking and bound to remote branches)
6. add - stage file for tracking (file **can** be changed)
7. commit - create a file snapshot (file **cannot** be changed, SHA is created and maintained all other information is also stored – email, change,…)
8. status - show untracked and staged files
9. branch - current branch and other local branches in repo (branch is a reference and nothing more)
10. merge - collaborate with mates and resolve conflicts
11. pull - download changes and branches from remote repo to sync
12. push - persist changes in remote repo

### Repo (three ways to work)

1. Create a new repo: git init my-repo (and etc.)
   1. git init
   2. git add README.md
   3. git commit -m "first commit"
      1. git remote add origin <https://github.com/rusergei2010/GITLAB_v2.git> (it will create a reference from origin -> remote repo)
   4. git push -u origin master (commit branch to repo associated with origin reference)
   5. It can ask authentication.
2. Fork existing repo and work solely
3. **Contribute into existing repo:**

# download a repository on GitHub.com to our machine

git clone https://github.com/me/repo.git

# change into the `repo` directory

cd repo

# create a new branch to store any new changes (**feature** branch)

**git branch my-branch**

# switch to that branch (line of development)

git checkout my-branch

# list branches

git branch -a

#remove branch

git branch -d my-branch

# make changes, for example, edit `file1.md` and `file2.md` using the text editor

# stage the changed files

git add file1.md file2.md

# stage all files

git add ./\*

git reset file\_path

# take a snapshot of the staging area (anything that's been added)

git commit -m "my snapshot"

# push changes to github

git push --set-upstream origin my-branch

1. There are 2 possible protocols: HTTPS/SSH to select in github

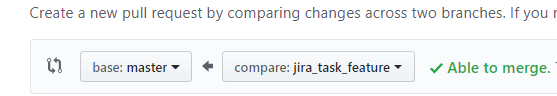
# Practical Part

## 1. Create your own project and branch

1. Go to git and create a blank project in gitlab. Explain what is gitlab.
2. Git clone, create branch, list branches, check out feature branch, modify ReadMe, add changes, commit changes, push, create merge request to master
   1. git clone <your\_project\_https>
   2. Modify ReadMe while in master
   3. git branch feature\_branch (changes will be persisted in a new branch)
   4. git branch (list branches)
   5. git checkout feature\_branch
   6. Add a new sample file
   7. git add new\_file
   8. git status (see all files - tracked, untracked, added)
   9. git push --set-upstream origin jira\_task\_feature (git push and see log)
   10. What is origin (it is a reference to remote repo - alias)

<https://stackoverflow.com/questions/9529497/what-is-origin-in-git>

* 1. Create Merge Request in gitlab
  2. 

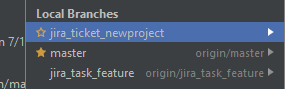


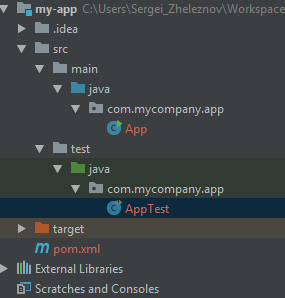
 & Confirm

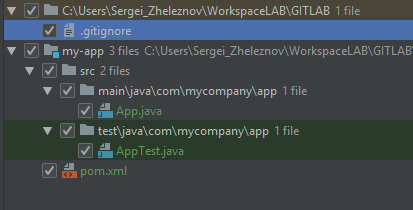
1. Generate a new Maven project, tracking files and adding under control, .gitignore .Generate Maven project (copy the command and execute):

*mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false*

1. cd ~/WorkspaceLAB/GITLAB/Lecture\_1\_16082018/my-app
   1. mvn install
   2. ./target dir is generated but it should be ignored
2. Open project in intellij and show the structure (open and set src/test/resource/testresoruce in module settings if it ismissing)
3. Create a new branch in Intellij:





1. Project can be put into VCS with bash or intellij IDE
2. 
3. .gitignore
4. 
5. Go to gitlab, create Merge Request and commit

## 2. Useful commands

1. git log --pretty=oneline(show commits)
2. git checkout <hash\_code> (detached branch)
3. git tag (list all tags)
4. git rebase (alternative to merge)
5. git show -2 (detailed info about 3 last commits)
6. **git remote show origin** (useful command to see all changes and branches)
7. git branch -r
8. git branch -v (show last commit on all branches)
9. git branch -d branch\_delete
10. git fetch origin (lookup remote repo/branch associated with origin like origin/master and update local version moving reference to the last version)
11. git push origin branch\_name (share without others)
12. git push origin master (will push trackable master branch and commits)
13. git fetch origin (will load remote branches, just information and references to them, it doesn’t create copies, to create a copy and start working then perform: git checkout -b feature\_remote\_branch origin/feature\_remote\_branch)
14. git checkout -b branch (crate a new one and checkit out - make track of it)
15. git push origin --delete feature\_branch\_been\_merged (remote remote branch that is been merged to master already)
16. git remote add pb <https://github.com/paulboone/ticgit> (add a new remote repo, check all trackable remote repos -> git remote -v)

## 3. Resolving conflicts

1. git fetch <origin> (laod changes from the remote repo, doesn’t merge with local changes)
2. git pull = git fetch & git merge
3. git reset file\_name (discard changes in the local repo being Added)

## 4. Tags

There are lightweight and usual tags.

Are creatd at a new relase version.

> git tag - listsall tags

$ git tag -l 'v1.8.5\*' - list with pattern to filter prefix release  
v1.8.5  
v1.8.5-rc0  
v1.8.5-rc1

> git push origin v1.8.5

> git push origin --tags (push all tags)

5. Revert changes:

git checkout -- file\_name (remove file)

git reset-- file\_name (remove from staging area added to git files)

git reset HEAD^ (remove two last commits: HEAD~20 - if 20 remove)

3. git revert (creates commit for revert)

git merge --abort

git checkout --ours / --theirs

git diff

Or use merge tool

To stash the changes:

git stash save “description”

git stash list

Bring them back:

git stash pop

git stash apply

Drop stash content: git stash drop

git tag release1.0

git tag --list

git push --tags

git checkout release1.0

--

## **Practice.**

Example:

1. Scenario 1 (assume there is already one commit with some file, add another)

git touch file (not tracked)

git add . (become trackable, if change then it is acceptable – not snapshot)

git commit -m “” (in local repo)

git reset –hard HEAD^1 (or the hash of the commit to remain)

git reset –soft HEAD^1 (commit will be reverted and file will be like just added)

git reset (will unstage file – revert add operation and file become untracked)

git clean -fd (remove untracked files)

2. Scenario 2 (push to remote repo)

git remote add origin <https://github.com/rusergei2010/GIT_test.git>  
git remote remove origin (remove reference,origin – reference to remote repo - alias)

git remote -v (see refs)

git push -u origin master (the remote repo need to be created beforehand)

* see remote repo on git

1. Scenario 3 (branches)

git branch JIRA-1\_feature\_branch

git checkout JIRA-1\_feature\_beanch

Add new file with content

git commit -m “new file added”

git push --set-upstream origin JIRA-1\_feature\_branch (keep the reference at commit time)

git checkout master

git merge JIRA-1\_feature\_branch (merge changes to the branch)- fast forward (then something in a feature branch is ahead)

1. Scenario 4

vim file.txt (add “changes 1 in master”)

git add .

git commit -m “changes in master”

git checkout JIR-1\_feature\_branch

vim file.txt (add “changes in JIRA feature”)

git add .

git commit -m “Added to feature”

git checkout master

git push (only master branch will be sync)

git push –-all (add branches will be in sync)

git merge JIRA-1\_feature\_branch (GIT GUI – necessary, but can be manually with vim)

vim file.txt (resolve it)

git commit -m “merge with JIRA-1\_feaure”

git push

git checkout JIRA-1\_feaure

git merge master (fast-forward)

1. Create Merge Request in GihHub for multiple merge commits

10. Create a new repo preserving history of commits from a branch of another repo:

git branch --set-upstream-to=origin/<branch> master

git push https://github.com/accountname/new\_repo.git +old\_branch:master