

**Ministerul Educației al Republicii Moldova**  
**Universitatea Tehnică a Moldovei**  
**Facultatea Calculatoare, Informatică și Microelectronică**  
**Departamentul Ingineria Software și Automatica**

# Raport

**Lucrare de laborator nr. 1**  
**la disciplina: *Programarea în rețea***  
***Tema: Versionarea codului sursă utilizând GIT***

**A efectuat:** st. gr. TI-142 Comanda Artur  
**A verificat:** lector asistent Ostapenco Stepan

## **Chișinău 2017**

### **Obiectivele lucrării**

Crearea unui repozitoriu distant, localizat de serviciul github, și sincronizarea tuturor modificărilor efectuate asupra repozitoriului local.

### **Sarcina lucrării**

Lucrarea de laborator are ca scop studiul și înțelegerea principiilor de funcționare și utilizare a sistemului distribuit de control al versiunilor numit GIT, utilizând repozitoriul creat- <https://github.com/irmate210/gitutm>

## GIT

Sistemele de versionare (VCS, Version Control Systems - eng.) servesc la gestionarea versiunilor multiple ale fișierelor incluse într-un proiect colaborativ. Fiecare modificare efectuată asupra elementului de proiect se memorizează împreună cu autorul schimbării. Important de menționat că în orice moment de timp se poate reveni la o versiune anterioară a entității.

Motivația principală constă în posibilitatea ca diferiți membri ai echipei, aflați eventual în spații geografice îndepărtate, să poată lucra simultan la proiect, urmând ca, la final, modificările lor să fie reunite în noi versiuni ale proiectului. De asemenea, există și alte avantaje. Când se observă un bug, se poate reveni la o versiune anterioară, în vederea determinării momentului introducerii acestuia în program. În același timp, se poate urma o dezvoltare pe ramuri (branches), în care se lucrează, în paralel, la multiple versiuni ale proiectului - de exemplu, una în care se dorește înlăturarea bug-urilor, iar cealaltă, în care se urmărește adăugarea de noi funcționalități, înaintea șlefuirii celor existente.

Există două modele de VCS-uri.

Modelul centralizat (ex: SVN): codul sursă este situat pe un server central, de unde clienții pot obține variante de lucru pe mașina locală (*working copy*). După efectuarea locală a modificărilor, dezvoltatorul solicită actualizarea variantei de pe server.

Avantajele utilizării modelului SVN sunt portabilitatea și suportul de integrare cu numeroase IDE-uri ca, de exemplu, Eclipse.

Principalele dezavantaje ale utilizării modelului centralizat SVN sunt:

- dependența de accesul la server;
- mentenanța și backup-urile serverului;
- dificultăți de comunicare cu serverul;
- dificultatea de a utiliza instrumentele pentru gestionarea ramurilor și de unificare a ramurilor.

Modelul distribuit (ex: GIT): nu există un server central, procesul de sincronizare desfășurându-se la nivel peer-to-peer.

Principalele beneficii ale sistemelor GIT și Mercurial sunt:

- urmărirea schimbărilor mai avansată și mai detaliată, lucru care duce la mai puține conflicte;
- lipsa necesității unui server, toate operațiile cu excepția schimbului de informații între repozitorii se realizează local;
- operațiile pentru gestionarea ramurilor și de unificare a ramurilor (prin intermediul comenzii merge) sunt

mai sigure, și prin urmare folosite mai des;

- rapiditatea marea a operațiilor datorită lipsei necesității comunicării cu serverul.

Dezavantajele utilizării sistemelor GIT și Mercurial:

- modelul distribuit este mai greu de înțeles; - nu există așa de mulți clienți GUI datorită faptului că aceste sisteme sunt mai noi;

- reviziile nu sunt numere incrementale, lucru care le face mai greu de referențiat;

- riscul apariției de greșeli este mare dacă modelul nu este familiar.

## **Chei SSH**

SSH este un protocol de rețea care asigură o comunicare securizată a datelor între două stații (calculatoare, tablete, telefoane sau alte dispozitive dintr-o rețea). Pentru a asigura confidențialitatea și integritatea informațiilor interschimbate, SSH se folosește de criptarea cu chei asimetrice. Criptografia asimetrică este un tip de criptografie care utilizează o pereche de chei: o cheie publică și o cheie privată. Un utilizator care deține o astfel de pereche își publică cheia publică astfel încât oricine dorește să o poată folosi pentru a îi transmite un mesaj criptat. Numai deținătorul cheii secrete (private) este cel care poate decripta mesajul astfel criptat. Utilizatorul deține o pereche de chei: una publică și una privată. În timp ce cheia publică se trimite stației de la distanță (eng. *remote*) cu care se dorește comunicarea, cea privată rămâne tot timpul pe stația locală și trebuie protejată de public. Cheia publică trebuie trimisă stației de la distanță, pentru că aceasta să o poată folosi la decriptarea datelor primite în format securizat.

## Mersul lucrării

În această lucrare de laborator pentru realizarea sarcinilor am utilizat sistemul de versionare de tip distribuit și anume Git.



Figura 1 – Logo-ul Git

Pentru început am instalat Git Bash. După ce am instalat componentele necesare și am setat cele necesare am început realizarea lucrării de laborator. Am creat o cheie SSH după care am setat-o pe website-ul github.com în setările contului personal. Modul de creare și setare a cheiei SSH poate fi vizualizată în Figura 1. Deoarece eu aveam cheia ssh generată pentru această mașină în directoriul .ssh unde fișierul id\_rsa.pub conține însăși cheia ssh cu email-ul setat, eu doar am adăugat-o în setările contului.

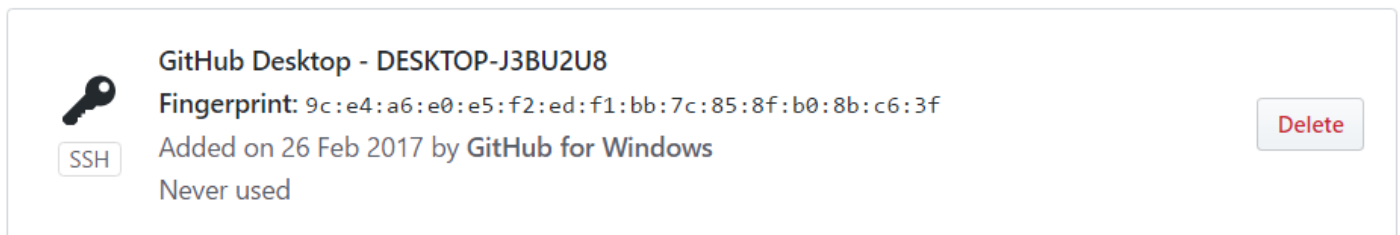


Figura 2 – Setarea cheiei SSH în setările contului de pe github.com

După ce am setat cheia SSH avem nevoie de creat un directoriu ce va fi repositoryul local de unde vom gestiona repositoryul distant și de aceea am creat un directoriu nou.

Am creat un directoriu local cu numele Git pentru realizarea un repository local pentru această lucrare de laborator. După creare am accesat acest directoriu prin utilizarea altei comenzi cd(change directory) și respectiv numele directoriului. În acest directoriu am și inițializat repositoryul. (Figura 3)

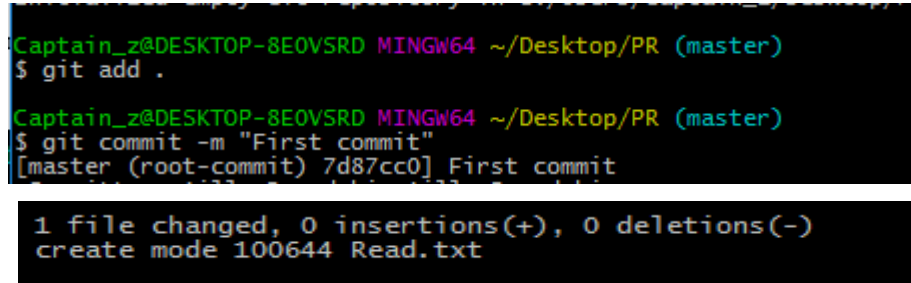
```
Captain_z@DESKTOP-8E0VSRD MINGW64 ~ (master)
$ cd Desktop

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop (master)
$ cd PR

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git init
Initialized empty Git repository in C:/Users/Captain_z/Desktop/PR/.git/
```

Figura 3 – Inițializarea repozitoriului local

În figura de mai sus este reprezentat modul de inițializare unui repozitoriu gol Git din Git Bash prin utilizarea comenzii `git init`. Astfel avînd un repozitoriu creat am adăugat cîteva fișiere și le-am adăugat pe repozitoriul distant cu mesajul respectiv care se face prin commit-uri. Aceast pas poate fi vizualizt în Figura 4.



```
Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git add .

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git commit -m "First commit"
[master (root-commit) 7d87cc0] First commit

1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 Read.txt
```

Figura 4 – Adăugarea unor fișiere

În figura de mai sus am adăugat fișierele dar deoarece acest repozitoriu este nou creat trebuie pentru început de realizat legătura cu repozitoriul distant și de aceea am utilizat comand `git remote` a cărui utilizare poate fi vizualizată în Figura 5.



```
Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git remote add origin https://github.com/Aillyrored/GitPR.git
```

Figura 5 – Realizarea legăturii cu repozitoriul distant

După ce am făcut legătura cu repozitoriul distant acum putem încărca fișierele adăugate mai sus și marcate cu mesajul respectiv pe server cu comand `git push -u origin master`. Astfel are loc transmiterea modificărilor în repozitoriul distant. (Figura 6)

```

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 207 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/Aillyrored/GitPR.git
* [new branch]      master -> master
Branch master set up to track remote branch master from origin.

```

Figura 6 – Transmiterea pe server a modificărilor

În figura de mai sus este reprezentat modul de încărcare a fișierilor și modificărilor pe repoziatoriul distant. După orice pas îndeplinit putem vizualiza starea repoziatoriului precum în Figura 7. Astfel în orice moment putem analiza starea repoziatoriului și efectuarea modificărilor respective pentru repoziatoriul distant.

```

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean

```

Figura 7 – Vizualizarea stării repoziatoriului

Am creat un nou repozioriu pentru încărcarea unor fișiere. Branch-urile sunt utilizate pentru a efectua modificări importante sau experimentale în afara proiectului stabile. În cazul dat am creat un branch conform sarcinii din lucrarea de laborator, un branch cu numele dev. (Figura 8)

```

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git branch dev

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git checkout dev
Switched to branch 'dev'

```

Figura 8 – Crearea unui branch și mutarea pe acesta

În figura dată am creat un nou branch și am trecut pe acesta dar această ooperație se poate de simplificat doar printr-o singură comandă `git branch -b checkout` și numele branch-ului. După ce am creat branch-ul am modificat ramura prin plasarea a unui fișier pe repozioriu cu mesajul respectiv și transmiterea modifcărilor pe server. (Figura 9)

```

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (dev)
$ git add ReadME.txt

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (dev)
$ git status
On branch dev
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   ReadME.txt

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (dev)
$ git commit -m "First dev"
[dev 2d8d367] First dev
Committer: Ailly Roredshi <Ailly Roredshi>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

    git config --global --edit

After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ReadME.txt

```

Figura 9 – Modificarea branch-ului dev

În cazul dat am adăugat un simplu fișier pe ramură și am atașat un mesaj pentru această modificare și transmiterea modificărilor pe server cu originea dev. După ce am modificat ramura dev am trecut din nou pe master și am adăugat din nou un fișier pe master cu mesajul respectiv. Și după aceasta am făcut concatenarea ramurilor master și dev. (Figura 10)

```

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (dev)
$ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git add ReadMaster.txt

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git commit -m "Second on Master"
[master d89c057] Second on Master

1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ReadMaster.txt

Captain_z@DESKTOP-8E0VSRD MINGW64 ~/Desktop/PR (master)
$ git merge dev
Merge made by the 'recursive' strategy.
 ReadME.txt | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ReadME.txt

```



Figura 10 – Git Merge

După efectuarea sarcinilor de laborator am obținut un arbore al modificărilor pe repository care este reprezentat în Figura 11. Astfel putem vizualiza grafic starea și modificările efectuate asupra repositoryului utilizând Git în IntelliJ.

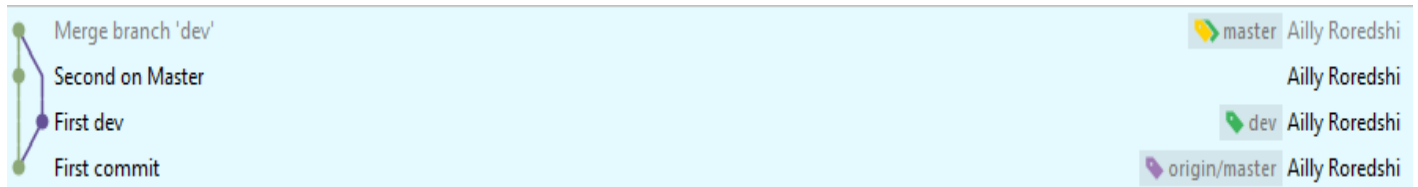


Figura 11 - Vizualizarea în IntelliJ a arborelui

Pentru partea a doua a lucrării de laborator am creat un proiect de tip maven în IntelliJ cu setările necesare ale sistemului unde am configurat JDK și Apache maven pentru ca sistemul să le recunoască. După aceasta în fișierul generat xml unde am configurat setările necesare ale proiectului : versiunea, url, dependențele și versiunea acesteia. În Figura 12 putem vizualiza deja fișierul xml cu toate configurațiile.

```

<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>PR_Lab1</groupId>
  <artifactId>PR</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>PR_1</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-math3</artifactId>
      <version>3.6.1</version>
    </dependency>
  </dependencies>
</project>

```

Figura 12 – Configurarea și specificarea setărilor proiectului

Iar pentru utilizarea dependenței commons-math3 am creat o clasă java cu numele Main.java în care am utilizat dependența și am importat-o în acest fișier. În metoda main am creat două numere de tip Complex cu datele necesare după care am făcut adunarea acestor numere, afișarea lor și rezultatul operațiilor efectuate. Acest lucru poate fi vizualizat în Figura 13.

```

import org.apache.commons.math3.complex.Complex;

public class PR {

    public static void main (String arg[])
    {
        Complex n1=new Complex( real: 5.15, imaginary: 9.99);
        Complex n2=new Complex( real: 5.05, imaginary: 8.18);
        Complex answer =n1.add(n2);
        System.out.println("Sum = " + answer);
    }
}

```

Figura 13 – Implementarea unei programe ce utilizează dependența

În figura de mai sus este reprezentat codul sursă a programului ce utilizează dependența definită în fișierul xml. Am utilizat din dependență clasa Complex și am efectuat o simplă operație de adunare între numerele definite. Rezultatul rulării este reprezentat în Figura 14.

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...  
Sum = (10.2, 18.17)  
  
Process finished with exit code 0
```

Figura 14 – Rularea programului

## Concluzie

În această lucrare de laborator am studiat un sistem de versionare a codului și anume GIT. Acest sistem este un mod de eficient de gestionare a codului sursă ce permite posibilitatea de a păstra istoricul tuturor modificărilor pe repozitoriu. Unul dintre avantajele folosirii sistemului GIT este posibilitatea de a reveni la o versiune a fișierelor mai vechi. Utilizatorul dispune de o copie de siguranță a codului sursă pe repozitoriul local. Cea mai recentă sau ultima modificată versiune a proiectului va fi disponibilă pe repozitoriul distant și poate fi accesat de toți dezvoltatori în caz că repozitoriul este public. La fel avem posibilitatea de a crea repozitorii private unde specificăm cine are acces asupra proiectului. GIT simplifică o problemă majoră în dezvoltare software și anume colaborarea dezvoltatorilor.

## **Bibliografie**

1. Instrumente pentru versionare [Resursă electronică] – Regim de acces:  
<http://andrei.clubcisco.ro/cursuri/f/fsym/4idp/elearn/13.%20Intrumente%20pentru%20Versionare.pdf>
2. Scott Chacon, Pro Git [Resursă electronica] – Regim de acces: <http://git-scm.com/book>