Facultatea Calculatoare, Informatica si Microelectronica

Universitatea Tehnica a Moldovei

Medii Interactive de Dezvoltare a Produselor Soft

Lucrarea de laborator#1

Version Control Systems si modul de setare a unui server

Autor:
Daniela Cazac

lector asistent:
Irina Cojanu
lector superior:
Radu Melnic

Lucrarea de laborator #1

1 Scopul lucrarii de laborator :

Studierea, familiarizarea su utilizarea unui Version Control System si modul de setare a unui server.

2 Obiective

Studierea Version Control System (git).

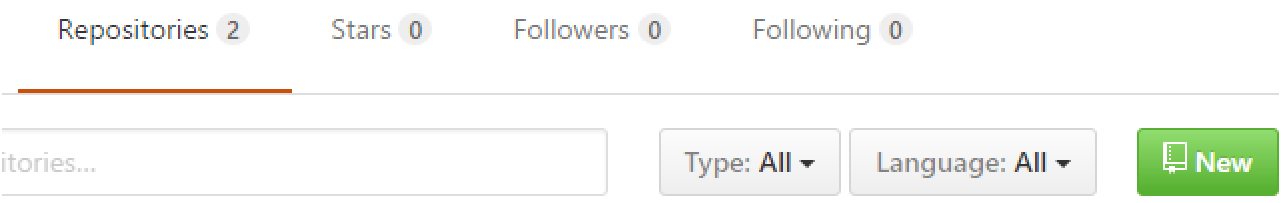
3 Mersul lucrarii de laborator

3.1 Cerinte :

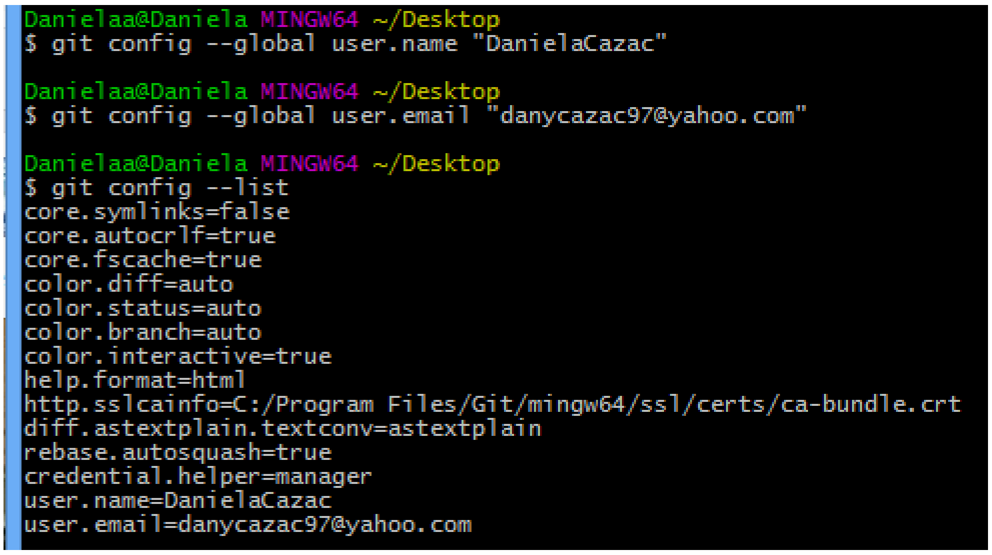
Initializerea unui nou repositoriu
Configurarea VCS
Crearea a 2 branch-uri
Commit pe ambele branch-uri (cel putin 1 commit per branch)
Setarea unui branch to track a remote origin pe care vei putea sa faci push
Reseteaza un branch la commit-ul anterior
Salvarea temporara a schimbarilor care nu se vor face commit imediat.
Folosirea fisierului .gitignore
Merge la 2 branch-uri
Rezolvarea conflictelor a 2 branch-uri

3.2 Analiza lucrarii de laborator

Linkul la repozitoriu este **https://github.com/daniela-cazac/MIDPS**
Pentru a **initializa un repozitoriu**, este necesar sa accesam site-ul : **https://github.com/** , iar repozitoriu poate fi creat fie odata cu crearea contului , fie avind un cont existent. Pentru a crea un repozitoriu pe un cont deja creat urmam pasii : **Repositories - New -** type **Repository Name -** tick **Initialize the repository with a README - Create repository** .

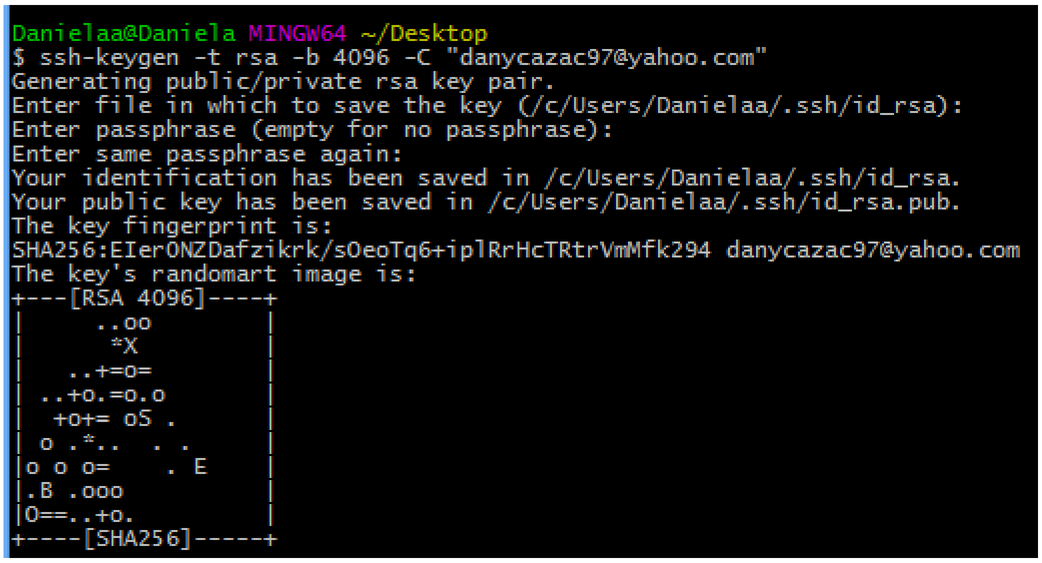


Pentru **configurarea Git-ului**, este necesar sa configuram numele si adresa email. Astfel prin comenzile : **git config -global user.name Name** si **git config -global user.email Email** configuram datele noastre, iar pentru a verifica daca acestea au fost validate scriem comanda **git config -list** care ne afiseaza o serie de informatii despre configurarile existente, astfel observam in ultimile 2 rinduri informatia introdusa de noi.



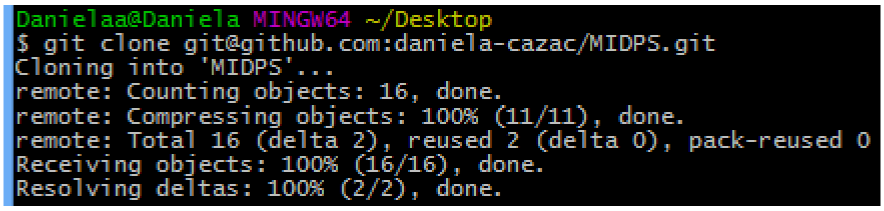
```
Danielaa@Daniela MINGW64 ~/Desktop
$ git config --global user.name "DanielaCazac"

Danielaa@Daniela MINGW64 ~/Desktop
$ git config --global user.email "danycazac97@yahoo.com"

Danielaa@Daniela MINGW64 ~/Desktop
$ git config --list
core.symlinks=false
core.autocrlf=true
core.fscache=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
help.format=html
http.sslcainfo=C:/Program Files/Git/mingw64/ssl/certs/ca-bundle.crt
diff.astextplain.textconv=astextplain
rebase.autosquash=true
credential.helper=manager
user.name=DanielaCazac
user.email=danycazac97@yahoo.com
```

Urmatorul pas este de a ne conecta la GitHub folosind o **cheie SSH**. Initial generam cheia prin introducerea comenzii : **ssh-keygen -t rsa -b 4096 -C ”your-email@example.com”** , apoi cheia obtinuta o copiem , si in Setarile profilului GitHub cream o cheie in care introducem cheia generata.



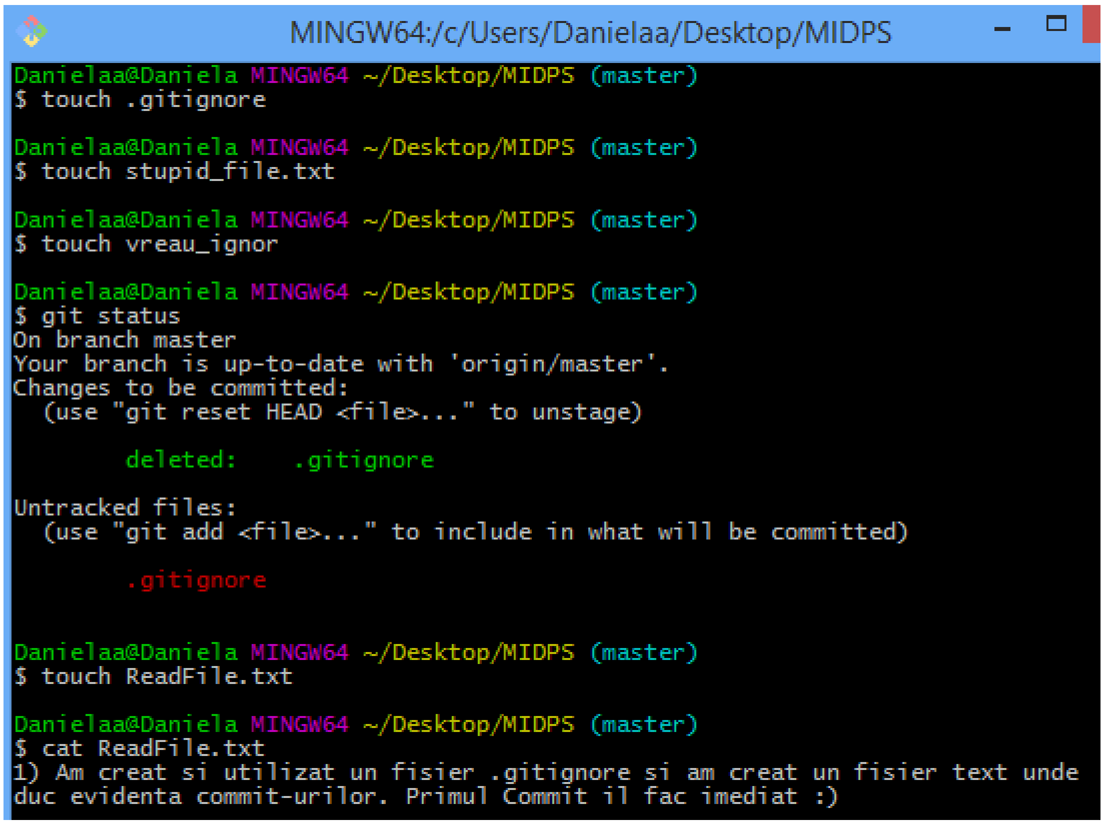
```
Danielaa@Daniela MINGW64 ~/Desktop
$ ssh-keygen -t rsa -b 4096 -C "danycazac97@yahoo.com"
Generating public/private rsa key pair.
Enter file in which to save the key (/c/Users/Danielaa/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /c/Users/Danielaa/.ssh/id_rsa.
Your public key has been saved in /c/Users/Danielaa/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:EIerONZDafzikrk/sOeoTq6+iplRrHcTRtrVmMfk294 danycazac97@yahoo.com
The key's randomart image is:
+---[RSA 4096]----+
|     ..oo        |
|      *X         |
|   ..+=o=        |
| ..+o.=o.o       |
|  +o+= oS .      |
| o .*..  . .     |
|o o o=    . E    |
|.B .ooo          |
|O==..+o.         |
+----[SHA256]-----+
```

Urmatorul pas am clonat repositoriul. Aceasta insemana ca se creaza local pe calculator o copie a repositorului. Pentru aceasta folosim comanda **git clone SSH URL** (copiat din sectiunea **Clone or download** de pe GitHub).



```
Danielaa@Daniela MINGW64 ~/Desktop
$ git clone git@github.com:daniela-cazac/MIDPS.git
Cloning into 'MIDPS'...
remote: Counting objects: 16, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 16 (delta 2), reused 2 (delta 0), pack-reused 0
Receiving objects: 100% (16/16), done.
Resolving deltas: 100% (2/2), done.
```

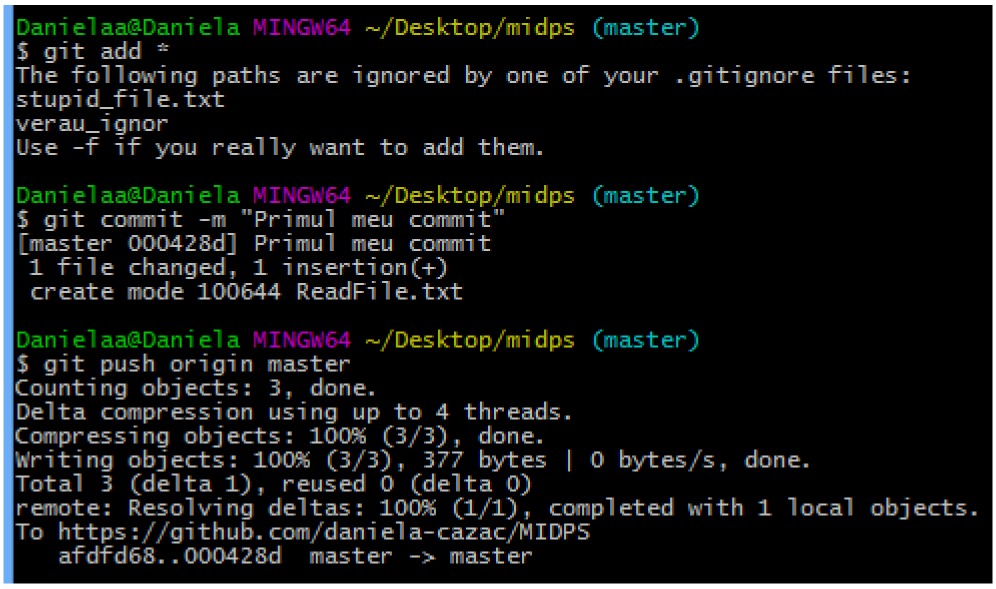
Pentru a **folosi fisierul .gitignore** trebuie sa-l cream in repozitoriu, iar Git il foloseste pentru a determina care fisiere si directorii sa le ignore inainte de a face commit. Am creat si niste fisiere pe care .gitignore le va ignora, pentru a demonstra astfel functionalitatea acestuia. Am creat inca un fisier unde voi duce evidenta commit-urilor , si care il voi modifica cu informatia necesara inaintea fiecarui commit. Comezii folosite:
touch .gitignore - pentru fisierului fara denumire ce va avea extensia gitignore iar in acest fisier vom adauga fisierele/directoarele noastre pe care nu le vrem urmarite
git status - pentru a verifica daca un fisier urmarit a fost modificat , i s-au adus modificari in continutul acestuia



MINGW64:/c/Users/Danielaa/Desktop/MIDPS

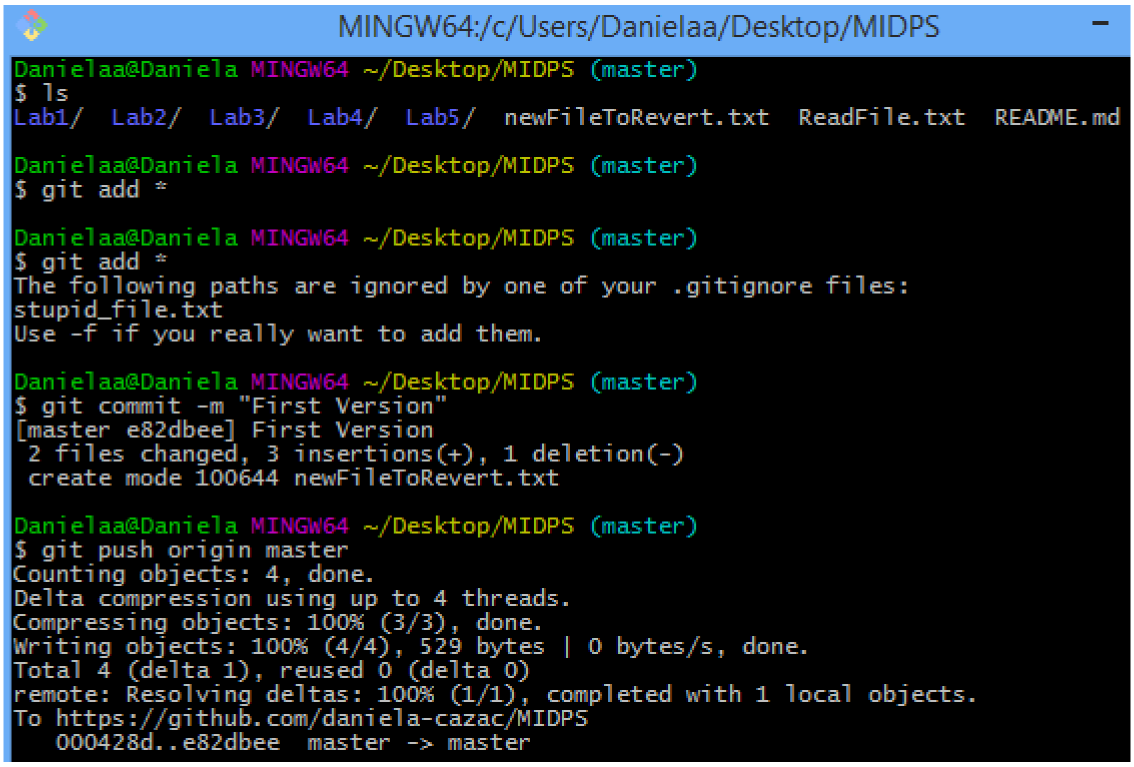
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ touch .gitignore

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ touch stupid_file.txt

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ touch vreau_ignor

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    .gitignore

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore


Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ touch ReadFile.txt

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ cat ReadFile.txt
1) Am creat si utilizat un fisier .gitignore si am creat un fisier text unde
duc evidenta commit-urilor. Primul Commit il fac imediat :)
```

Urmatorul pas , dupa crearea fisierelor, este sa facem primul commit utilizind : **git add .** - adauga toate fisierele cu continut text si nu numai ce se afla in directoare , pentru a fi inregistrate (caracterul punct == toate)
git commit -m - salveaza toate modificarile aduse la fisierele noastre
git push origin master - incarca toate modificarile pe **http://github.com**

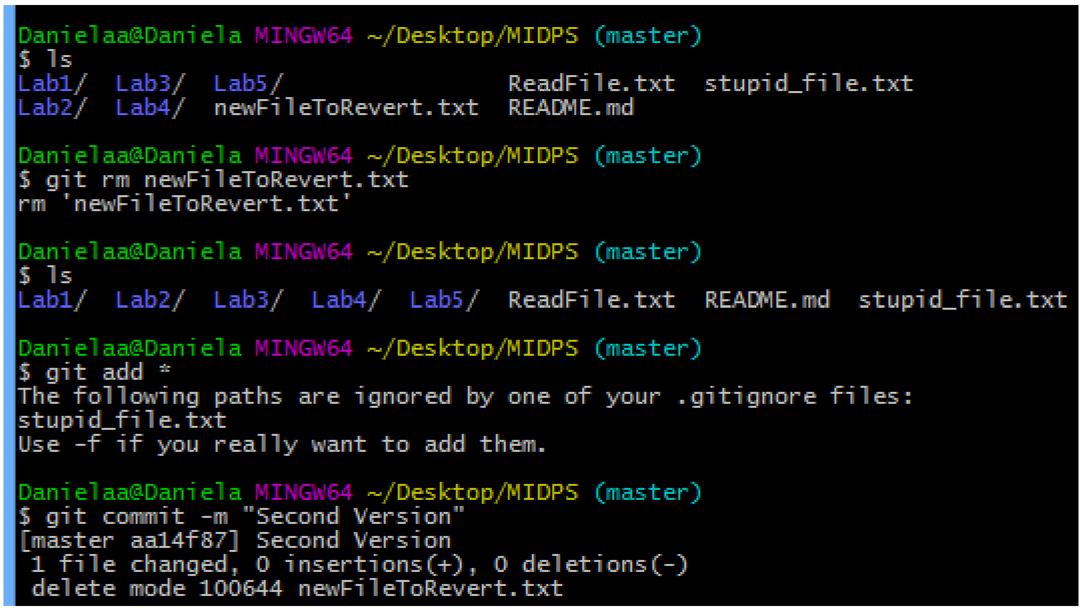


```
Danielaa@Daniela MINGW64 ~/Desktop/midps (master)
$ git add *
The following paths are ignored by one of your .gitignore files:
stupid_file.txt
verau_ignor
Use -f if you really want to add them.

Danielaa@Daniela MINGW64 ~/Desktop/midps (master)
$ git commit -m "Primul meu commit"
[master 000428d] Primul meu commit
 1 file changed, 1 insertion(+)
 create mode 100644 ReadFile.txt

Danielaa@Daniela MINGW64 ~/Desktop/midps (master)
$ git push origin master
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 377 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/daniela-cazac/MIDPS
   afdfd68..000428d  master -> master
```

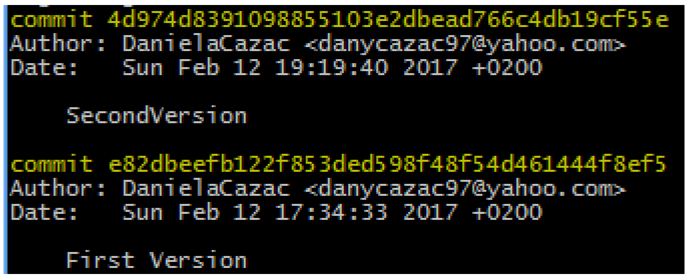
Pentru a demonostra **resetarea unui commit anterior** voi crea 2 versiuni : **FirstVersion** si **SecondVersion**. In prima voi include fisierul **newFileToRevert.txt** si voi face commit pe Git, iar intr-a doua versiune,voi sterge acel fisier si din nou commit!
Apoi utilizind comanda **git log** putem vedea codul,autorul si timpul fiecarui commit facut. Vom copia primele 7 cifre ale codului pe care le vom introduce in urmatoarea comanda **git reset –hard** *******
Ulterior vom observa un mesaj care ne informeaza ca acum suntem la FirstVersion cu codul indicat anterior.
PEntru verificare vom introduce comanda **ls** pentru a ne afisa lista de fisiere. Astfel vom vedea ca fisierul care a fost sters in versiunea a doua va aparea din nou,ceea ce demonstreaza ca am revenit la prima versiune cu siguranta!



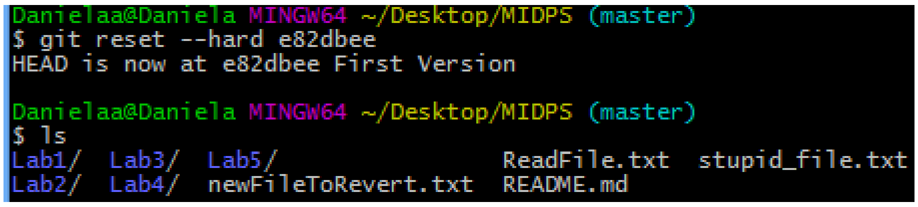
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ ls
Lab1/  Lab2/  Lab3/  Lab4/  Lab5/  newFileToRevert.txt  ReadFile.txt  README.md

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git add *

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git add *
The following paths are ignored by one of your .gitignore files:
stupid_file.txt
Use -f if you really want to add them.

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git commit -m "First Version"
[master e82dbee] First Version
 2 files changed, 3 insertions(+), 1 deletion(-)
 create mode 100644 newFileToRevert.txt

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 529 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/daniela-cazac/MIDPS
   000428d..e82dbee  master -> master
```



```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ ls
Lab1/  Lab3/  Lab5/                    ReadFile.txt  stupid_file.txt
Lab2/  Lab4/  newFileToRevert.txt  README.md

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git rm newFileToRevert.txt
rm 'newFileToRevert.txt'

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ ls
Lab1/  Lab2/  Lab3/  Lab4/  Lab5/  ReadFile.txt  README.md  stupid_file.txt

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git add *
The following paths are ignored by one of your .gitignore files:
stupid_file.txt
Use -f if you really want to add them.

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git commit -m "Second Version"
[master aa14f87] Second Version
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 newFileToRevert.txt
```

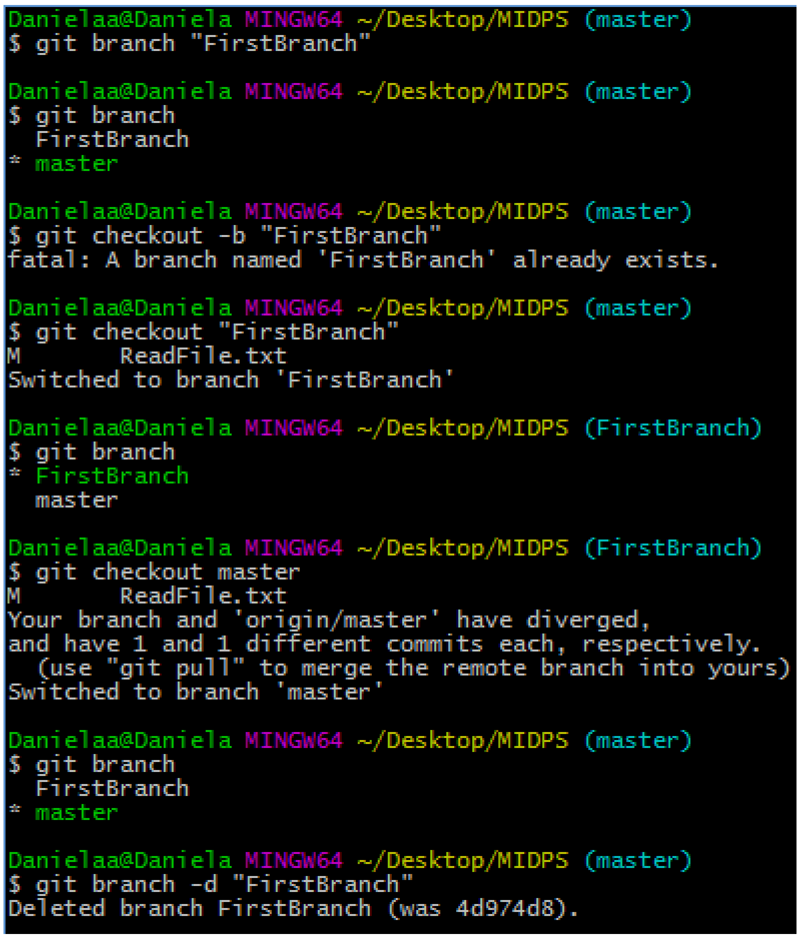


```
commit 4d974d8391098855103e2dbead766c4db19cf55e
Author: DanielaCazac <danycazac97@yahoo.com>
Date:   Sun Feb 12 19:19:40 2017 +0200

    SecondVersion

commit e82dbeefb122f853ded598f48f54d461444f8ef5
Author: DanielaCazac <danycazac97@yahoo.com>
Date:   Sun Feb 12 17:34:33 2017 +0200

    First Version
```



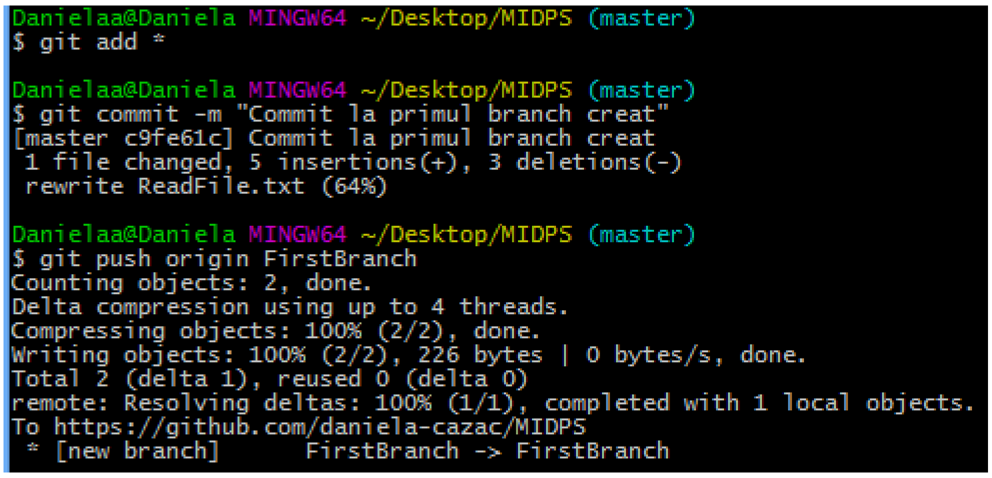
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git reset --hard e82dbee
HEAD is now at e82dbee First Version

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ ls
Lab1/  Lab3/  Lab5/                    ReadFile.txt  stupid_file.txt
Lab2/  Lab4/  newFileToRevert.txt  README.md
```

Un branch este o ramura independenta de dezvoltare.Avantajul este ca permite izolarea lucrului de lucrul celalorlalti membrii ai echipei.
Crearea unui **branch** se face cu ajutorul comenzii **git branch "NameOfTheBranch"**.
In mod implicit , branch-ul Git-ului este **Master**. Astfel, la creearea unui nou branch , si cind le vom afisa, vom avea 2 branc-uri : cel implicit si cel nou creat. In mod implicit, lucreaza Master, dar cind avem nevoie sa lucram pe alta ramura atunci facem switch cu comanda **git checkout -b "name"**.
In continuare am demonstrat crearea,aratarea listei de ramuri , trecerea de la o ramura la alta si stergerea.



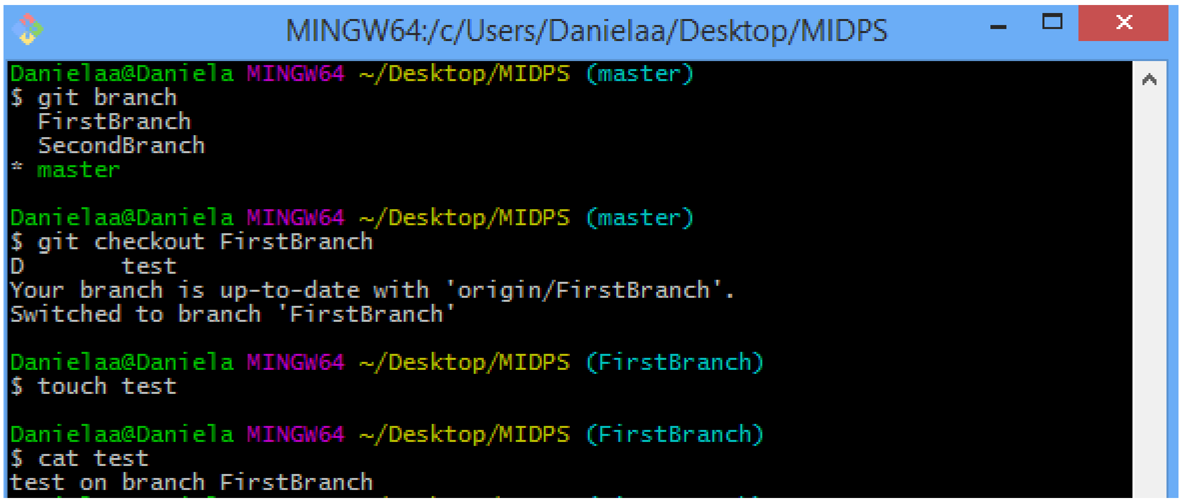
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git branch "FirstBranch"

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git branch
  FirstBranch
* master

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git checkout -b "FirstBranch"
fatal: A branch named 'FirstBranch' already exists.

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git checkout "FirstBranch"
M       ReadFile.txt
Switched to branch 'FirstBranch'

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ git branch
* FirstBranch
  master

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ git checkout master
M       ReadFile.txt
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
Switched to branch 'master'

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git branch
  FirstBranch
* master

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git branch -d "FirstBranch"
Deleted branch FirstBranch (was 4d974d8).
```

Facem commit la primul branch creat :



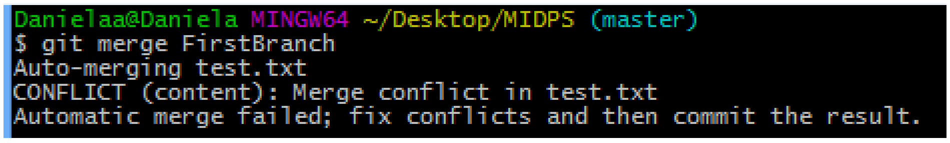
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git add *

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git commit -m "Commit la primul branch creat"
[master c9fe61c] Commit la primul branch creat
 1 file changed, 5 insertions(+), 3 deletions(-)
 rewrite ReadFile.txt (64%)

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git push origin FirstBranch
Counting objects: 2, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (2/2), 226 bytes | 0 bytes/s, done.
Total 2 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To https://github.com/daniela-cazac/MIDPS
 * [new branch]      FirstBranch -> FirstBranch
```

Cu comanda **git branch** aratam branch-urile disponibile. Cu comanda **git checkout "nameOfTheBranch"** am facut switch pe branch-ul repsectiv. In branch-ul curent in care ne aflam am creat un fisier ¡test¿ in care am scris ¡test on branch FirstBranch¿. Acesti pasi i-am prezentat in urmatorul screenshoot:



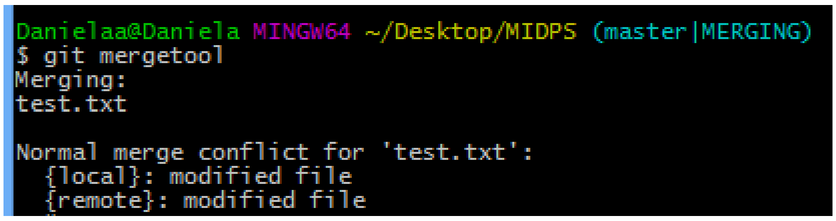
```
MINGW64:/c/Users/Danielaa/Desktop/MIDPS
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git branch
  FirstBranch
  SecondBranch
* master

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git checkout FirstBranch
D       test
Your branch is up-to-date with 'origin/FirstBranch'.
Switched to branch 'FirstBranch'

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ touch test

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ cat test
test on branch FirstBranch
```

Al doilea branch cu vom face conflic este Master , astfel in el am creat la fel un fisier test in care am scris **test on branch 2**. In continuare , scriem comanda **git merge NameOfTheBranch** , si prin urmare obtinem conflict din cazua diferentelor care le-am creat intentionat.



```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git merge FirstBranch
Auto-merging test.txt
CONFLICT (content): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
```

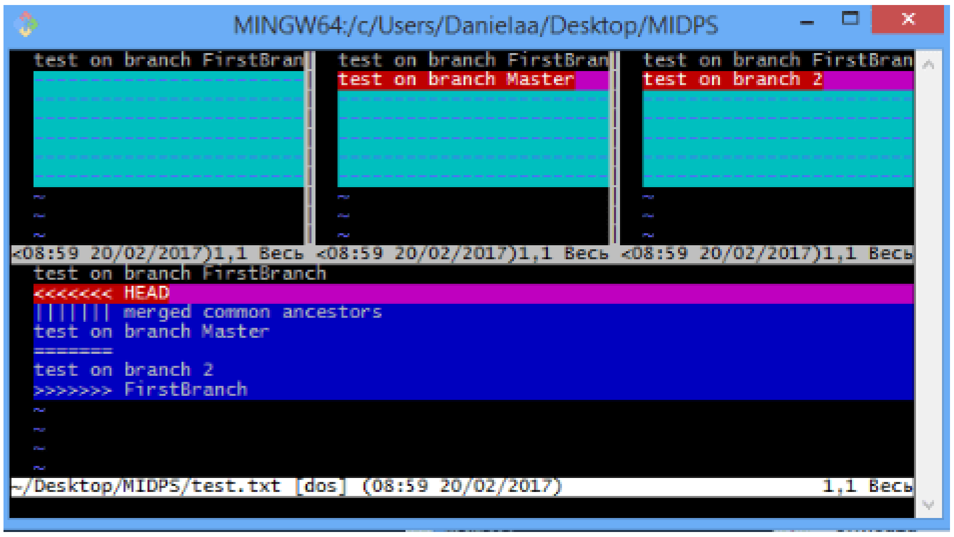
Acum comanda **git mergetool** ne arata care fisier are conflict si urmeaza a fi merge-uit :D Intradevar fisierul **test.txt** este cu pricina.



```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master|MERGING)
$ git mergetool
Merging:
test.txt

Normal merge conflict for 'test.txt':
  {local}: modified file
  {remote}: modified file
```

Pentru a rezolva conflictul am folosit un tool , si anume **vdiff** care a fost sugerat de insusi GitBush, care implicit era configurat. Dupa ce am scris comanda **git mergetool** automat se deschide alta fereastra cu 3 sectiuni: -prima sectiune **Local** este un fisier temporar care afiseaza continutul din branch-ul curent.
-a doua sectiune **Base** este un fisier temporar care afiseaza baza comuna pentru a rezolva conflictul.
-a treia sectiune **Base** este un fisier temporar care afiseaza continutul fisierului care urmeaza a fi merge-uit.

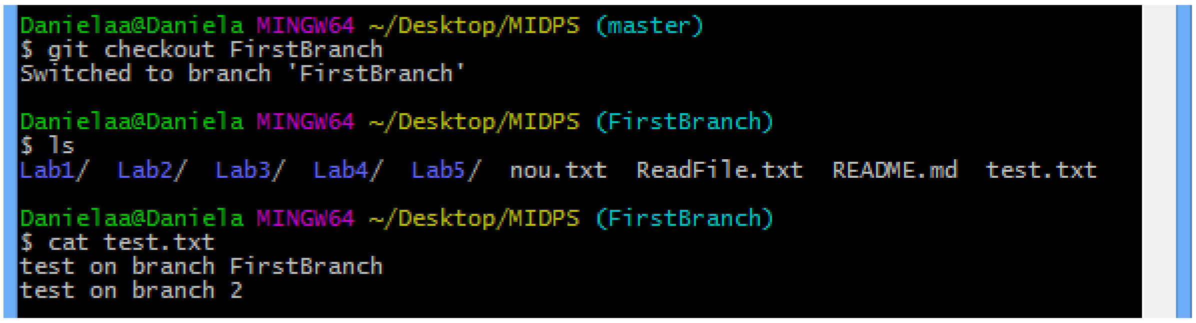
Iar a patra sectiune **Merged** este un fisier care afiseaza conflictele. In acest fisier stergem liniile deprisos si lasam doar acele linii care le dorim : astfel eu am ales sa las:
test on branch FirstBranch
test on branch 2
test on branch Master



MINGW64:/c/Users/Danielaa/Desktop/MIDPS

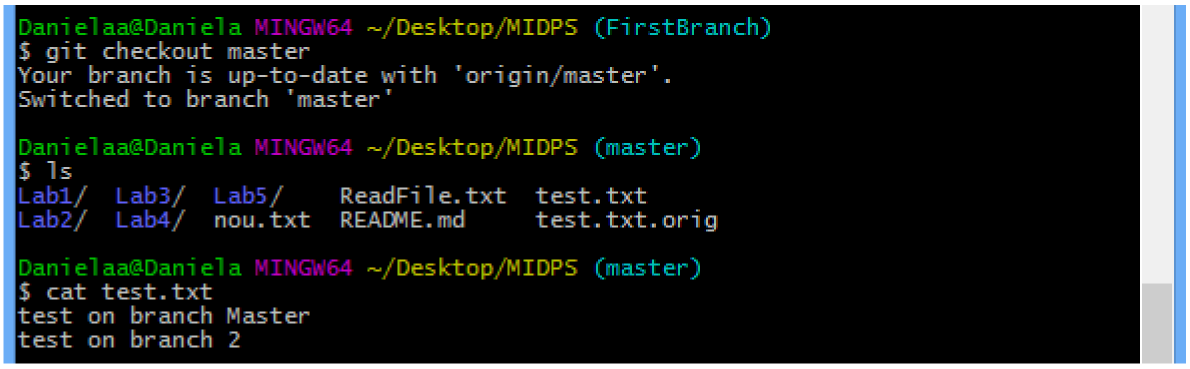
```
test on branch FirstBran   test on branch FirstBran   test on branch FirstBran
                           test on branch Master      test on branch 2
~                          ~                          ~
~                          ~                          ~
~                          ~                          ~
<08:59 20/02/2017)1,1 Весь <08:59 20/02/2017)1,1 Весь <08:59 20/02/2017)1,1 Весь
test on branch FirstBranch
<<<<<<< HEAD
||||||| merged common ancestors
test on branch Master
=======
test on branch 2
>>>>>>> FirstBranch
~
~
~
~
~/Desktop/MIDPS/test.txt [dos] (08:59 20/02/2017)                       1,1 Весь
```

Astfel am rezolvat conflictul. Eu am ales sa rezolv acest conflict prin combinare : pe branch-ul Master cind voi afisa continutul fisierului voi avea:
test on branch Master
test on branch 2



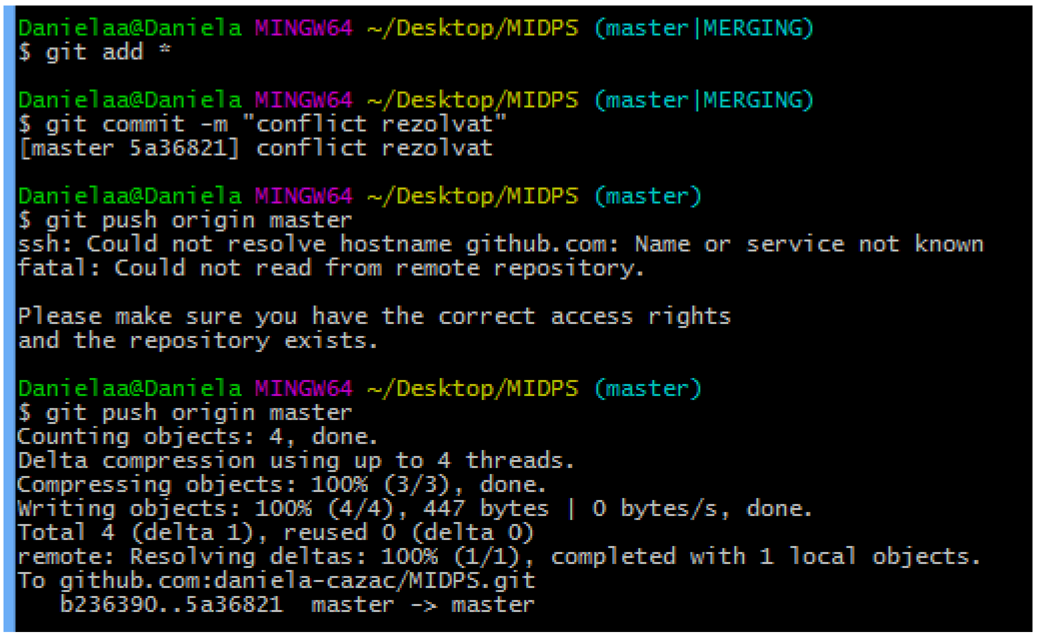
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git checkout FirstBranch
Switched to branch 'FirstBranch'

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ ls
Lab1/  Lab2/  Lab3/  Lab4/  Lab5/  nou.txt  ReadFile.txt  README.md  test.txt

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ cat test.txt
test on branch FirstBranch
test on branch 2
```

Iar cind voi afisa continutul fisierului de pe branch-ul FirstBranch voi avea :
test on branch FirstBranch
test on branch 2



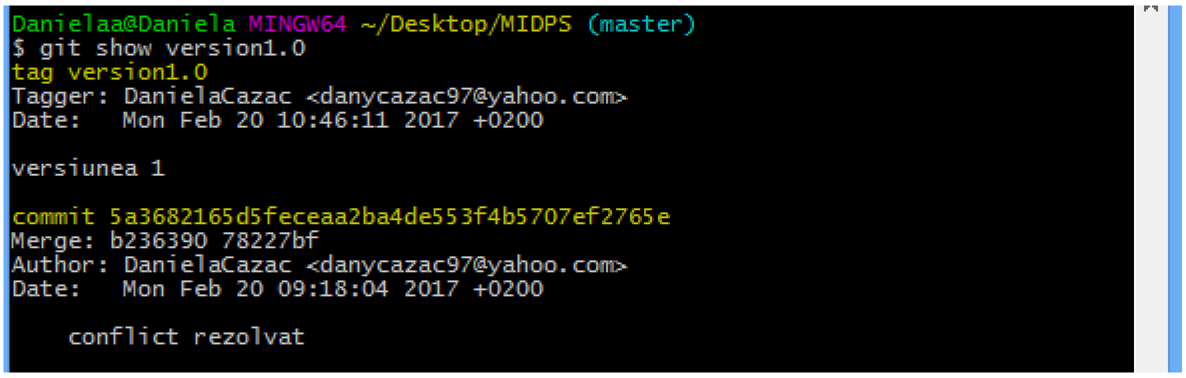
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (FirstBranch)
$ git checkout master
Your branch is up-to-date with 'origin/master'.
Switched to branch 'master'

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ ls
Lab1/  Lab3/  Lab5/    ReadFile.txt  test.txt
Lab2/  Lab4/  nou.txt  README.md     test.txt.orig

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ cat test.txt
test on branch Master
test on branch 2
```

Si am facut commit-ul final :



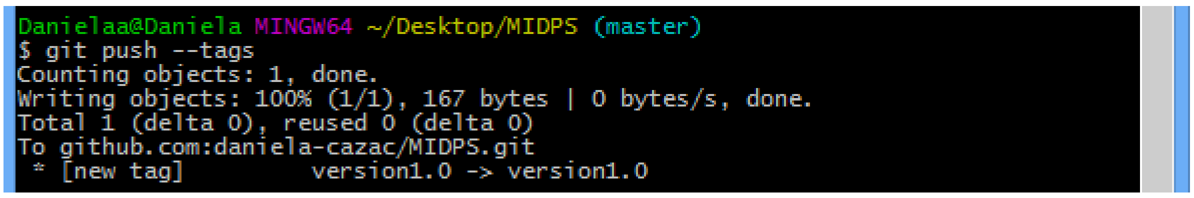
```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master|MERGING)
$ git add *

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master|MERGING)
$ git commit -m "conflict rezolvat"
[master 5a36821] conflict rezolvat

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git push origin master
ssh: Could not resolve hostname github.com: Name or service not known
fatal: Could not read from remote repository.

Please make sure you have the correct access rights
and the repository exists.

Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git push origin master
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 447 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To github.com:daniela-cazac/MIDPS.git
   b236390..5a36821  master -> master
```

Tag-urile in Git reprezinta o metoda eficienta de a organiza mai bine lucrul nostru. Astfel putem salva sub forba de versiuni.
Cu comanda **git tag -a version1.0** am creat un tag.
Iar cu comanda **git show NameOfTheTag** ne afiseaza versiunile disponibile si informatii aditionale.



```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git show version1.0
tag version1.0
Tagger: DanielaCazac <danycazac97@yahoo.com>
Date:   Mon Feb 20 10:46:11 2017 +0200

versiunea 1

commit 5a3682165d5feceaa2ba4de553f4b5707ef2765e
Merge: b236390 78227bf
Author: DanielaCazac <danycazac97@yahoo.com>
Date:   Mon Feb 20 09:18:04 2017 +0200

    conflict rezolvat
```

Si cu comanda **git push –tags-** facem transferul tag-urilor de pe local pe site.



```
Danielaa@Daniela MINGW64 ~/Desktop/MIDPS (master)
$ git push --tags
Counting objects: 1, done.
Writing objects: 100% (1/1), 167 bytes | 0 bytes/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To github.com:daniela-cazac/MIDPS.git
 * [new tag]         version1.0 -> version1.0
```

4 Concluzie

In aceasta lucrare de laborator am facut cunostinta cu Version Control System care este o categorie de software tool-uri care ajuta o echipa de a organiza schimbarile codului in orice moment de timp. Version Control System pastreaza track-urile oricarei modificari, de aceea un avantaj al sau este eficienta, posibilitatea dezvoltatorilor de a lucra in paralel mai rapid. Asfel o echipa de dezvoltatori pot lucra individual la partea sa de lucru , dar la final sa le uneasca.De asemenea ofera posibilitatea de a reveni la o versiune anterioara, daca cea curenta nu ne convine,am creat un repositoriu local in care am lucrat,am creat fisiere,le-am editat,am creat branch-uri,am rezolvat conflicte utilizind un tool anume,am creat tag-uri si am facut commit-uri. Comenzile de baza si cele mai des folosite le-am aplicat in aceasta lucrare.

Am contientizat ca cu ajutorul GIT-ului putem lucra mult mai usor si eficient intr-o echipa.