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:
Dumitrita Garaba
lector asistent:
Irina Cojanu

Lucrarea de laborator #1

0.1 Scopul lucrarii de laborator

Studierea principiilor de creare si de lucru cu Git .

0.2 Obiective

Version Control Systems (git || bitbucket || mercurial || svn).

0.3 Efectuarea lucrarii de laborator

0.3.1 Tasks and Points

Basic Level (nota 5 || 6) :
-initializeaza un nou repositoriu
-configureaza-ti VCS
-crearea branch-urilor (creeaza cel putin 2 branches)
-commit pe ambele branch-uri (cel putin 1 commit per branch)
Normal Level (nota 7 || 8):
-seteaza un branch to track a remote origin pe care vei putea sa faci push (ex. Github, Bitbucket or custom server)
-reseteaza un branch la commit-ul anterior
-salvarea temporara a schimbarilor care nu se vor face commit imediat.
-folosirea fisierului .gitignore
Advanced Level (nota 9 || 10):
-merge 2 branches
-rezolvarea conflictelor a 2 branches
-comezile git care trebuie cunoscute
Bonus Point (+1):
-Tags. Folosirea tag-urilor pentru marcarea schimbarilor simnificative precum release-ul.

0.3.2 Analiza lucrarii de laborator

Linkul la repozitoriu: `https://github.com/dumitritag/MIDPS-lab`

1.Initializarea unui repositoriu si configurarea VCS

Initial a fost creat un cont pe GitHub care contine repozitoriul laboratorului efectuat. Am creat repozitoriul meu *MIDPS−lab*. A fost stabilita conexiunea cu serverul prin generarea keygen-ului SSH prin instructiunea *ssh-keygen*, ea fiind adaugata in setari. Dupa ce am clonat repozitoriul, folosind comanda *git clone shhlik* , am configurat contul cu comenzile *git config −global user.name "YourName"* si *git config −global user.email "youremail@domain.com"* Am creat fisierele .gitignore si README in repozitoriu si cu ajutorul comenzii *git pull* le-am adaugat in masina locala. In masina locala am creat 5 mape, fiecare

continind cite un fiser README si le-am adaugat in repozitoriu cu comenzile *git add .* , *git commit* si *git push .* Commiturile sunt schimbarile salvate in poiectu nostru. Fiecare commit are un mesaj asociat, in care descrim ce schimbare am facut. In mapa Lab1 din masina locala am creat un fisier myfile.txt cu continutul "Welcome!" si l-am adaugat in repozitoriu.

2. Crearea branch-urilor

Branch-ul implicit in Git este master, comanda *git init* il creeaza implicit. Am creat 2 branch-uri noi: branch1 si branch2 cu ajutorul comenzii *git branch ¡branchname¿.* Aceasta creeaza un nou pointer la ultimul commit. Pentru a comuta la un branch nou am folosit comanda *git checkout ¡branchname¿.* Am modificat fisierul myfile.tx, l-am adaugat la branch-urile si am facut primele commituri in noile branch-uri.

3. Track remote origin

Am creat un nou branch newbranch to track remote origin, folosind comanda *git branch –track ¡branchname¿ origin/master.* Am creat un nou fisier new.txt, l-am adaugat, am facut commit si cu comanda *git push origin ¡branchname¿* am adaugat commitul in newbranch iar cu comanda *git push origin HEAD:master* in master branch.

4. Resetarea unui branch la commitul anterior

Pentru rescrierea commitului anterior am folosit comanda *git commit –amend.* Pentru resetarea unui branch la commitul anterior am folosit comanda *git revert HEAD.*

5. Salvarea temporara a schimbarilor

Uneori facem schimbari care nu dorim sa le facem commit dar dorim sa le salvam, asta putem face cu comanda *git stash.*

6. Folosirea fisierului .gitignore

Pentru a vedea cum influenteaza fisierul .gitignore, am adaugat in fisierul .gitignore *.txt si am creat un nou fisier de tip txt, am aplicat comanda git add si acest fisier a fost ignorat.

7. Merge 2 branches

Merge 2 branches inseamna a le integra intr-un singur branch. Pentru aceasta am folosit comanda *git merge ¡branchname¿.*

8. Rezolvarea conflictelor a 2 branch-uri

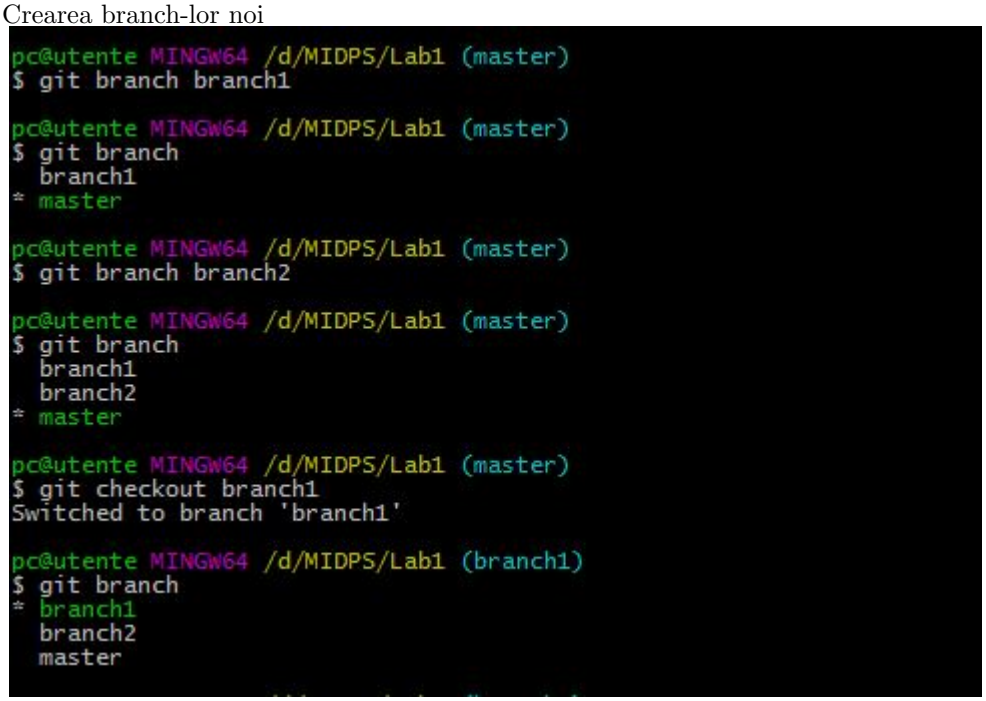
Daca avem 2 branches, in ambele a fost schimbat aceeasi parte a unui fisier si dorim sa facem merge la ambele va aparea un conflict. Acest conflict putem sa-l rezolvam manual, modificind fisierul care a cauzat conflictul. O alta metoda este prin folosirea comenzii *git rebase master.*

9. Folosirea tag-urilor

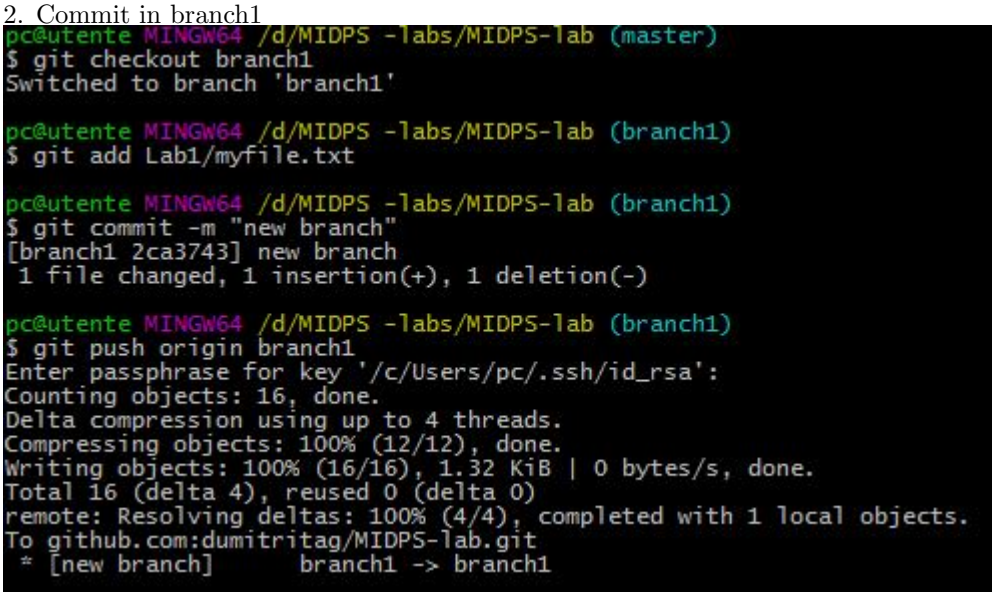
Un tag este folosit pentru a eticheta si a marca un commit. Sunt 2 tipuri de tag-uri lightweight, folosim comanda *git tag ¡tagname¿* si annotated, comanda folosita este *git tag -am "message" ¡tagname¿.*

0.3.3 Imagini

1. Crearea branch-lor noi



2. Commit in branch1



3. Commit in branch 2

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch1)
$ git checkout branch2
Switched to branch 'branch2'

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2)
$ git add Lab1/myfile.txt

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2)
$ git commit -m "branch 2"
[branch2 2a3aa75] branch 2
 1 file changed, 2 insertions(+), 1 deletion(-)

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2)
$ git push origin branch2
Enter passphrase for key '/c/Users/pc/.ssh/id_rsa':
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 363 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:dumitritag/MIDPS-lab.git
 * [new branch]      branch2 -> branch2
```

4. Track remote origin

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git branch --track newbranch origin/master
Branch newbranch set up to track remote branch master from origin.

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git remote show origin
Enter passphrase for key '/c/Users/pc/.ssh/id_rsa':
* remote origin
  Fetch URL: git@github.com:dumitritag/MIDPS-lab.git
  Push  URL: git@github.com:dumitritag/MIDPS-lab.git
  HEAD branch: master
  Remote branches:
    branch1 tracked
    branch2 tracked
    master  tracked
  Local branches configured for 'git pull':
    master    merges with remote master
    newbranch merges with remote master
  Local refs configured for 'git push':
    branch1 pushes to branch1 (up to date)
    branch2 pushes to branch2 (up to date)
    master  pushes to master  (up to date)
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git checkout newbranch
Your branch is up-to-date with 'origin/master'.
Switched to branch 'newbranch'

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (newbranch)
$ git add Lab1/new.txt

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (newbranch)
$ git commit -m "commit on newbranch"
[newbranch 3fc08ef] commit on newbranch
 1 file changed, 1 insertion(+)
 create mode 100644 Lab1/new.txt
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (newbranch)
$ git push origin HEAD:master
Enter passphrase for key '/c/Users/pc/.ssh/id_rsa':
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 377 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:dumitritag/MIDPS-lab.git
   9e9a7e3..3fc08ef  HEAD -> master

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (newbranch)
$ git push origin newbranch
Enter passphrase for key '/c/Users/pc/.ssh/id_rsa':
Total 0 (delta 0), reused 0 (delta 0)
To github.com:dumitritag/MIDPS-lab.git
 * [new branch]      newbranch -> newbranch
```

5. Redenumirea unui commit

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (newbranch)
$ git add Lab1/new.txt

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (newbranch)
$ git commit --amend
[newbranch c763dc9] first commit on newbranch
 Date: Wed Feb 22 13:14:46 2017 +0100
 1 file changed, 2 insertions(+)
 create mode 100644 Lab1/new.txt
```

```
commit on newbranch

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Wed Feb 22 13:14:46 2017 +0100
#
# On branch newbranch
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#       new file:   Lab1/new.txt
#
```

6. Resetarea la commitul anterior

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git log
commit 80d2e2fc75258f83d21961754ae68716b3857109
Author: Garaba Dumitrita <dumitritag1@gmail.com>
Date:   Sun Feb 26 16:43:31 2017 +0100

    change new txt file
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git revert HEAD
[master fb11060] Revert "change new txt file"
 1 file changed, 1 insertion(+), 1 deletion(-)
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git log
commit fb1106041c62634c2c3b9a82f762ead4f0d590f9
Author: Garaba Dumitrita <dumitritag1@gmail.com>
Date:   Sun Feb 26 16:46:31 2017 +0100

    Revert "change new txt file"

    This reverts commit 80d2e2fc75258f83d21961754ae68716b3857109.
```

7. Salvarea temporara a schimbarilor

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git stash
Saved working directory and index state WIP on master: 9e9a7e3 first commit
HEAD is now at 9e9a7e3 first commit
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

8. Folosirea fisierului .gitignore

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git add dumi.txt
The following paths are ignored by one of your .gitignore files:
dumi.txt
Use -f if you really want to add them.
```

9. Merge 2 branches

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git merge branch1
Merge made by the 'recursive' strategy.
 Lab1/myfile.txt | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git merge branch2
Auto-merging Lab1/myfile.txt
CONFLICT (content): Merge conflict in Lab1/myfile.txt
Automatic merge failed; fix conflicts and then commit the result.
```

10. Rezolvarea conflictelor
Manual:

```
Welcome!
<<<<<<< HEAD
How are you?!
=======
Have a nice day!
>>>>>>> branch2
```

```
Welcome!
How are you?!
Have a nice day!
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master|MERGING)
$ git add Lab1/myfile.txt

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master|MERGING)
$ git commit -m "merge branch2 "
[master 2fdb243] merge branch2
```

Comanda git rebase:

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git reset --hard HEAD~
HEAD is now at 51b804f Merge branch 'branch1'

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git checkout branch2
Switched to branch 'branch2'

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2)
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: branch 2
Using index info to reconstruct a base tree...
M       Lab1/myfile.txt
Falling back to patching base and 3-way merge...
Auto-merging Lab1/myfile.txt
CONFLICT (content): Merge conflict in Lab1/myfile.txt
Patch failed at 0001 branch 2
The copy of the patch that failed is found in: .git/rebase-apply/patch

When you have resolved this problem, run "git rebase --continue".
If you prefer to skip this patch, run "git rebase --skip" instead.
To check out the original branch and stop rebasing, run "git rebase --abort".

error: Failed to merge in the changes.
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2|REBASE 1/1)
$ git add Lab1/myfile.txt

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2|REBASE 1/1)
$ git rebase --continue
Applying: branch 2
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (branch2)
$ git checkout master
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
  (use "git pull" to update your local branch)
Switched to branch 'master'

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git merge branch2
Updating 51b804f..32d611a
Fast-forward
 Lab1/myfile.txt | 6 +++++-
 1 file changed, 5 insertions(+), 1 deletion(-)
```

11. Tags

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git tag MIDPS

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git tag
MIDPS
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git show MIDPS
commit daf923ca3adda85f5d39c8d900f67dfcb64f2236
Merge: 32d611a 2fdb243
Author: Garaba Dumitrita <dumitritag1@gmail.com>
Date:   Wed Feb 22 22:02:30 2017 +0100

    ok
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git tag -am "First lab" lab1

pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git tag -n
MIDPS           ok
lab1            First lab
```

```
pc@utente MINGW64 /d/MIDPS -labs/MIDPS-lab (master)
$ git push origin --tags
Enter passphrase for key '/c/Users/pc/.ssh/id_rsa':
Counting objects: 6, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 744 bytes | 0 bytes/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local objects.
To github.com:dumitritag/MIDPS-lab.git
 * [new tag]         MIDPS -> MIDPS
 * [new tag]         lab1 -> lab1
```

Concluzie

Un sistem de control al versiunilor reprezinta un produs software ce ajuta dezvoltatorii dintr-o echipa software sa colaboreze in cadrul diferitelor proiecte si de asemenea pastreaza un jurnal complet al muncii fiecaruia. Un sistem de control al versiunilor (SCV) are trei scopuri principale :

1. Sa ofere posibilitatea muncii simultane, nu seriale

2. Atunci cand mai multe persoane lucreaza in acelasi timp se asigura ca modificarile realizate de acestia nu intra in conflict unele cu altele

3. Sa ofere o arhiva a fiecarei versiuni continand informatii despre cine, unde si din ce motiv a fost facuta fiecare modificare.

Sunt 2 forme de organizarea a sistemelor de control al versiunilor: centralizata si distribuita.

Principiul de baza al sistemelor centralizate se bazeaza pe relatia client-server. Un depozit (repository) este situat intr-un singur loc iar mai multi clienti au acces la el. Toate modificarile utilizatorilor si toate informatiile legate de aceste modificari (utilizator, data, revizie) sunt transmise si preluate de la un depozit (repository) central.

Sistemele de control al versiunilor distribuite sunt o optiune mai noua. In cadrul acestora fiecare utilizator are propria copie a intregului repository, nu doar fisiere, ci intregul jurnal. Aceasta abordare foloseste modelul peer-to-peer spre deosebire de modelul client-server folosit de sistemele centralizate. In acest caz sincronizarea dintre repository-uri este realizata prin schimbul de changeset-uri sau patch-uri dintre statii. Doua dintre cele mai folosite SCV-uri de acest fel sunt Git si Mercurial.

Principalele beneficii ale sistemelor Git sunt :

- O urmarire a schimbarilor mai avansata si mai detaliata, lucru care duce la mai putine conflicte

- Lipsa necesitatii unui server-toate operatiile cu exceptia schimbului de informatii intre repository-uri se realizeaza local

- Operatiile de branching si merging sunt mai sigure, si prin urmare folosite mai des

- Rapiditate mai mare a operatiilor datorita lipsei necesitatii comunicarii cu serverul

Principalele dezavantaje ale sistemelor Git :

- Modelul distribuit este mai greu de inteles

- Nu exista asa de multi clienti GUI datorita faptului ca aceste sisteme sunt mai noi

- Reviziile nu sunt numere incrementale, lucru ce le face mai greu de referentiat

- Riscul aparitiei de greseli este mare daca modelul nu este familiar

Bibliography

[1] https://backlogtool.com/git-guide/en/stepup/stepup71.html

[2] https://www.siteground.com/tutorials/git/commands.htm

[3] https://git-scm.com/book/en/v2