

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:
Mărgineanu Cristian

lector asistent:
Irina Cojanu

lector superior:
Radu Melnic

Lucrare de laborator Nr.1

1 Scopul lucrarii de laborator

Invatarea unui Version Control System si a modului de setare a unui server

2 Obiective

Studierea a Version Control Sysrems (git)

3 Implimentarea lucrarii de laborator

3.1 Sarcini si Obiective

- initializeaza un nou repository
- configureaza-ti VCS
- crearea branch-urilor (creeaza cel putin 2 branches)
- commit pe ambele branch-uri (cel putin 1 commit per branch)
- 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.
- Folosire fisierului gitignore
- Merge 2 branches
- Rezolvare conflictelor a 2 branches
- Comenzile git care trebuies cunoscute
- Tags. Folosirea tag-urilor pentru marcarea schimbarilor semnificative pentru release-ul

3.2 Analiza lucrării de laborator

1. Basic Level

-initializarea un nou repository

La prima etapa cream un nou repository , urmind pasii din ghidul

de utilizare de pe platforma github, accesind butonul " + " si op-tiunea create new repository. Exista mai multe modalitati de a ini-tializa un repository pe github. Putem creea o mapa goala in care v-om plasa gitul nostru prin intermediul comenzii **git init** , crearea unui nou repository este posibil datorita comenzii **curl -u 'USER' https://api.github.com/user/repos -d '{"name": "NUME"}**

-configurarea VCS

Esential este configuram accountul git si repository. Esential este generarea cheii SSH(Secure Shell). Daca scriem in repository CLI **ssh-keygen**, iar cheia obtinuta o copiem in setarile noastre de pe git. nece-sar sa unim gitul nostru gol cu repository creat folosind comanda **git remote add origin "Linkul catre repository"** Pentru a clona re-pozitoriu in mapa locala de pe memoria calculatorului, utilizam linkul HTTP al acestuia, care se va copia de pe repository creat prin co-manda "Copy to clipboard".

Pentru aceasta ar trebui sa introducem in linia de comanda gitbash urmatoarele caractere:

git config --global user.name "YourName"

git config --global user.email "youremail@domain.com"

```
$ git config --global user.name "margineanu-cristian"
Dream House@DESKTOP-8032LE6 MINGW64 /e
$ git config --global user.email "cwcris1995@hotmail.com"
```

Putem adauga fsiere in repository cu ajutorul comenzii: **git add** si sa verifcam starea acestuia prin: **git status**

-crearea branch-urilor (creeaza cel putin 2 branches) Pentru a crea o ramura in linia de comanda se executa urmatoarele comenzi:

git checkout -b branch1 cream ramura

git branch -a vedem toate ramurile existente

Introducem schimbari in ramura respectiva;

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git branch first_branch
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git branch second_branch
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git branch
  first_branch
* master
  second_branch
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git checkout first_branch
Switched to branch 'first_branch'
```

-**commit pe ambele branch-uri** (cel puțin 1 commit per branch) Pentru a face commit se utilizează comanda:

git commit -am "new feature branch1" cream commit-ul

git push origin branch1 actualizăm și trimitem fișierul branch1 în repository de pe GitHub

git checkout master accesăm iarăși ramura inițială

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (first_branch)
$ git add .
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (first_branch)
$ git commit -m "First commit-first_branch"
[first_branch c6f43fe] First commit-first_branch
2 files changed, 5 insertions(+), 1 deletion(-)
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (first_branch)
$ git status
On branch first_branch
nothing to commit, working tree clean
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (first_branch)
$ git checkout second_branch
Switched to branch 'second_branch'
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (second_branch)
$ git status
On branch second_branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
```

```
    modified:   Lab 1/index.html
    modified:   Lab 1/style.css
```

```
no changes added to commit (use "git add" and/or "git commit -a")
```

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (second_branch)
$ git add .
```

2

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (second_branch)
$ git commit -m "First commit-second branch"
[second_branch 9dd001b] First commit-second branch
2 files changed, 6 insertions(+), 1 deletion(-)
```

2.Normal Level (nota7||8):

- **seteaza un branch to track a remote origin pe care vei putea sa faci push (ex. Github, Bitbucket or custom server)**

Pentru a seta ca o ramura sa urmareasca originul prin care am putea face push pe Github. Putem face track remote origin direct de pe ramura master **git push** sau putem crea alta ramura RemoteBranch si executa comanda **git push origin <RemoteBranch>**

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git push origin first_branch
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 534 bytes | 6.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To https://github.com/margineanu-cristian/MIDPS.git
 * [new branch]      first_branch -> first_branch
```

-reseteaza un branch la commit-ul anterior

Daca vrem sa ne intoarcem la un commit anterior, ramura noastra trebuie sa contina citeva commituri care pot introduce cu comenzile:

git add .

git commit -m "new feature branch1" cream commit-ul var.1

git commit -am "new feature branch1" cream commit-ul var.2

Astfel oricare commit anterior va putea resetat cu ajutorul a 3 tipuri de reseturi, in functie de scopul urmarit, insa intii si intii vom efectua comanda **git log** pentru a primi o lista de commituri cu codurile re-spectiv:

git reset --soft <codul h> care poate contine 6-9, pe care il vom copia din git log resetarea soft, sierele commitului pot accesate git **reset <codul h>** resetarea moderata, sierele se sterg insa sunt inca in working stage

git reset --hard <codul h> resetarea puternica, sterge toate sierele complet, pot desi niste siere untracked ramase daca ele au existat in commit, insa ele pot usor eliminate

git clean -df stergerea fiierelor neurmarite/untracked

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (experimental)
$ git log --oneline
711aa11 (HEAD -> experimental) Versiunea 2
edf64a5 Prima versiunea
7522665 (origin/master, origin/HEAD, master) Aceasta zi
49087b5 o zi frumoasa
de67227 (tag: v1.3) Simplu text
8eb3832 (tag: v1.2) First commit
HEAD is now at edf64a5 Prima versiunea
```

-salvarea temporara a schimbarilor care nu se vor face com-mit imediat.

Schimbarile pot salvate temporar. Aceasta este eficient atunci cind lucram asupra a mai multor taskuri. Avind mai multe fisiere neter-minate insa la care putem lucra in viitor. Nu este rezonabil sa facem commit la un

asemenea fisiier deoarece ceea la ce facem commit ar trebui sa fie executabil.

Respectiv avem nevoie de o modalitate prin care sa introducem modi-carile intro parte a memoriei si sa o accesam dupa ce am facut careva modificari in alta ramura si am hotarit sa continuam lucru asupra unei parti de program, respectiv utilizam un **stash**/un ascunzis

git stash save " Am lucrat asupra unui fisier" salvam in stash deja cind efectuam **git diff** facem diferenta dintre server si mapa locala

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (experimental)
$ git stash save "Ceva modificari nereusite"
stash/
Saved working directory and index state On experimental: Ceva modificari nereusi
te
```

git status verificam starea directoriului ramurii, care de regula ar trebui sa contina un fisier modificat de tip untracked dar cere nu con-tine nimic deja deoarece acesta a fost mutat in stash/ascunzis

Daca dupa anumit timp dorim sa accesam fisierul modificat, putem proceda astfel:

git stash pop accesam elementul din stash, si il autodistrugem

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (experimental)
$ git stash pop
On branch experimental
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   Lab 1/index.html

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (297c9f8278b1055b7d86c1dd83420c834cf54792)
```

-folosirea fisierului .gitignore

Un sier .gitignore poate creat direct de pe mapa locala MIDPS sau cu ajutorul comenzii **touch .gitignore**

.gitignore-ul este util pentru a specifica tipurile de fisier care vor neglijate/ignoreate.

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ touch .gitignore
```

3. Advanced Level (nota9jj10):

-merge 2 branches

git merge <denumire.ramura> facem merge la 2 ramuri

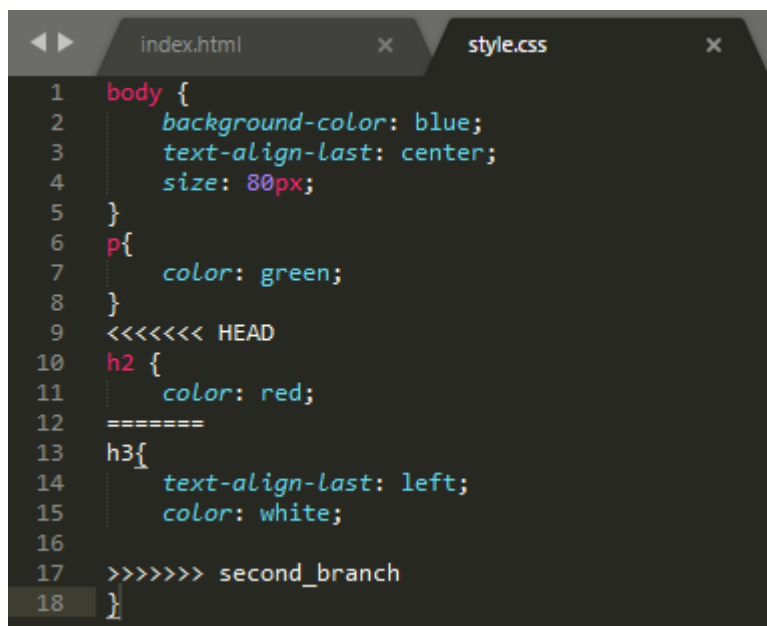
Se face merge/suprapunere a doua ramuri, daca ne aflam pe ramura1 si executam aceasta comanda git merge ramura2, toate modifcarile de pe ramura2 se vad in ramura1 nu si invers;

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (first_branch)
$ git merge second_branch
Auto-merging Lab 1/style.css
CONFLICT (content): Merge conflict in Lab 1/style.css
Auto-merging Lab 1/index.html
CONFLICT (content): Merge conflict in Lab 1/index.html
Automatic merge failed; fix conflicts and then commit the result.
```

-rezolvarea conflictelor a 2 branches

Atunci cind este un conflict cauzat de comanda merge, e datorat faptului ca un fisier se rescrie si apare in fisierul respectiv niste instructiuni pentru solutionarea conflictului.

Dupa ce corectam fisierul facem in acesta un commit si salvam acest fisier.



```
1  body {
2      background-color: blue;
3      text-align-last: center;
4      size: 80px;
5  }
6  p{
7      color: green;
8  }
9  <==== HEAD
10 h2 {
11     color: red;
12 }
13 h3{
14     text-align-last: left;
15     color: white;
16 }
17 >==== second_branch
18 }
```

comezile git care trebuie cunoscute

Comenzile git au fost explicate de mine pe parcursul tuturor pasilor insa alte comenzi utile care au servit sursa de inspiratie pot accesate aici: <https://orga.cat/posts/most-useful-git-commands>

4. Bonus point:

-Folosirea tag-urilor pentru marcarea schimbarilor semnificative precum release-ul

Taggingul se refera la crearea unor puncte specifice pentru repositoryul tau. Care sunt utilizate pentru a marca releasurile .

Este util pentru a marca o varianta executabila a unui program si permite notarea sub o forma speci ca ca:

V1.0

V1.1

Pentru a face un tag trebuie sa alegem o ramura/ un branch **git checkout <denumire.ramura>** si pasii principali sunt:

git tag V1.0 crearea simpla a unui tag

git tag -a V1.1 -m "nume tag" crearea anotata a unui tag Diferenta dintre aceste doua tipuri va ca crearea anotata va pastrata ca un obiect complet in git repository

Pentru a arata tag putem efectua comenzile: **git tag**
a asarea tuturor tag-urilor

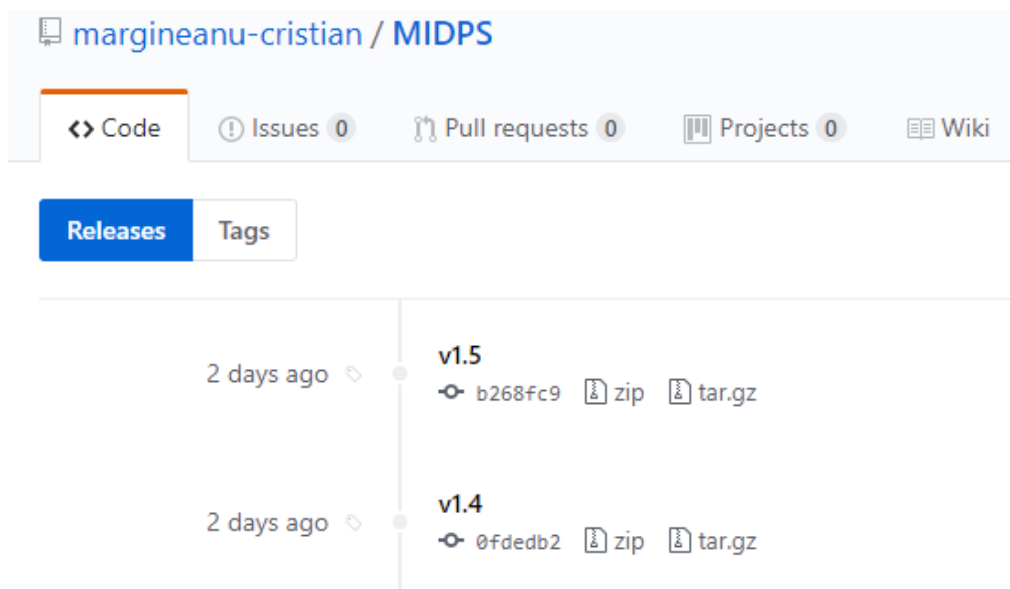
git show V1.0 afisarea sierului respectiv

git tag -l"V1.*" afisarea tuturor fisierelo care incep cu V1. pentru a crea un release in repository se foloseste:

git push origin V1.0

```
Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git push origin v1.5
Counting objects: 5, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 497 bytes | 23.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To https://github.com/margineanu-cristian/MIDPS.git
 * [new tag]          v1.5 -> v1.5

Dream House@DESKTOP-8032LE6 MINGW64 /e/midps (master)
$ git push origin v1.4
Counting objects: 1, done.
Writing objects: 100% (1/1), 163 bytes | 54.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To https://github.com/margineanu-cristian/MIDPS.git
 * [new tag]          v1.4 -> v1.4
```



margineanu-cristian / MIDPS

<> Code 0 Issues 0 Pull requests 0 Projects 0 Wiki

Releases Tags

2 days ago v1.5
b268fc9 zip tar.gz

2 days ago v1.4
0fdedb2 zip tar.gz

Concluzie

In urma realizarii acestei lucrari de laborator am reusit sa adaug cunostinte esentiale in ceea ce priveste platforma github. Crearea unui repository, asocierea acestui repository cu un fisier local, transmiterea prin comanda push a unor elemente in repository. A fost nou pentru mine crearea unor ramuri pe care putem pastra diferite variatii, extensii ale programului principal. Ramurile sunt extrem de usor de utilizat si accesat. Am invatat si m-am acomodat cu programul git bash, prin care m-am invatat sa fac anumite elemente precum, stash-uri pentru versiuni incipiente, tag-uri pentru a expune versiuni imbunatatite a programului principal. M-am familiarizat cu notiunea de commit. Am utilizat numeroase comenzi si am facut cunostinta cu extensia gitk. Deasemenea m-am familiarizat cu programa winmerge care am descoperit in timpul efectuarii acestei lucrari, programa care exclude conflictele de suprapunere a doua fisiere, fapt ce imi este de folos si in alte domenii. Pe parcursul elaborarii lucrarii am aflat ca un pull request se poate crea si din command line. Posibilitatile care le ofera git sunt foarte mari, insa unicul neajuns pe care l-am gasit eu este ca acesta opereaza cu foarte multe comenzi, si daca acestea nu sunt utilizate zi de zi, exista mereu necesitatea de a face search pe google in cautarea comenzii necesare.

Bibliografie:

<https://www.youtube.com/watch?v=PEKN8NtBDQ0>
<https://www.youtube.com/watch?v=9d5bJc8o7MA>
<https://www.youtube.com/watch?v=vFj2-bKGwkw>
<https://www.youtube.com/watch?v=ZFYhW3kBjnE>
<https://orga.cat/posts/most-useful-git-commands>