Facultatea Calculatoare, Informatică și Microelectronică

Universitatea Tehnică a Moldovei

Medii Interactive de Dezvoltare a Produselor Soft

Lucrarea de laborator#1

Version Control Systems și modul de setare a unui server

Autor:
Marina Jechiu

lector asistent:
Irina Cojanu

Laboratory work #1

1 Scopul lucrarii de laborator

Însusirea noțiunii de Version Control Systems și a modului de setare a unui server.

2 Obiective

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

3 Condiția sarcinii

– inițializează un nou repositoriu;

– configurează-ți VCS;

– crearea branch-urilor (cel puțin 2);

– commit pe ambele branch-uri;

– setează un branch to track a remote origin pe care vei putea să faci push ;

– salvarea temporară a schimbărilor care nu se vor face commit imediat;

– resetează un branch la commit-ul anterior;

– folosirea fișierului .gitignore;

– merge 2 branches;

– rezolvarea conflictelor a 2 branches.

4 Laboratory work implementation

4.1 Analiza lucrarii de laborator

https://github.com/MarinaJechiuTI154/MIDPS - linkul repozitoriului.
Principalele noțiuni cu care voi opera:

– **repository** componenta server ce conține informații privind ierarhia de fișiere și reviziile.

– **branch** este o ramură secundară de dezvoltare a unui proiect.

– **checkout** preluarea în mediul local a unei anumite revizii publicate pe server.

– **commit** cerere de publicare pe server a unor modificări.

– **pull** acțiunea de actualizare (update) a informațiilor locale cu cele de pe server.

– **conflict** apare atunci când mai mulți utilizatori vor să publice modificări aplicate acelorași fișiere din proiect, însă sistemul de aplicare a versiunilor diferite nu poate îmbina modificările.

– **revert** revenirea la o versiune anterioară pe un anume fir de dezvoltare (branch).

– **tag** branch "read-only" ce nu mai permite modificări ulterioare (folosit uneori pentru versiunile stabile și derivă dintr-un branch)

Am creat un cont public pe github cu denumirea mdps. Pentru a putea gestiona repozitoriul am instalat GitBash. Pentru a activa contul avem nevoie ss introducem în setări cheia, care se obține prin tastarea în linia de comandă: ssh-keygen. Pentru a deschide cheia obținută folosim următoarele instrucțiuni: cat /.ssh /id - rsa.pub. Pentru a face legătura între repozitoriul pe github și cel local este necesar să clonăm repozitoriul online cu instrucțiune: *git clone*.

La crearea repozitoriului se creează un branch implicit numit master. Însă, dacă trebuie să creăm un nou branch folosim comanda:*git checkout 'denumire-branch'*. Astfel, nu doar se creează un nou branch, dar și se trece automat pe acest branch. În cazul în care dorim să trecem pe un branch deja existend, la tastarea denumirii acestuia sistemul îl recunoaște și face automat salt către acesta.

Deoarece repozitoriile git sunt repozitorii de tip distrib, pentru a încărca modificările efectuate local pe server este necesar penrtu a efectua următoarii pași:

- *git add .* adăugarea în index a modificărilor realizate în directoriul meu, fișiere ce se intenționează a fi publicate.

  git commit efectuarea commit-urilor în baza informației din index. Acestea pot conține denumiri pentru a gestiona mai ușor modificările.

- *git push* publicarea modificărilor pe repozitoriu.

Pentru a putea face push pe branch-urile create, este necesar ca acestea să fie setate to track a remote origin. Pentru asta vom folosi comanda: *git push -u origin den-branch*.

Ultimul commit efectuat în repozitoriu poartă numele de HEAD. Astfel, pentru a reveni la un commit anterior este suficient să scriem în linia de comandă: *git reset –hard HEAD*.

Uneori, vrem să facem anumite schimbări temporare, dar fără a face deodat[commit asupra acestora. Pentru asta utilizăm comanda *git stash*. Putem totodată să revenim asupra acestor modificări mai tîrziu.

Nu întotdeauna dorim sa versionăm anumite fisiere, întrucât acestea pot conține executabile generate de procesul de compilare sau efectiv parametrii de configurare ai aplicației. Fișierele sau directoarele ce se doresc a fi ignorate de sistemul de versionare pot fi trecute într-un fișier numit .gitignore. Acesta este evaluat in mod recursiv, putand astfel avea mai multe fișiere .gitignore în folderele proiectului nostru. Cu toate acestea, pentru a nu căuta foarte mult path-urile ignorate, se folosește în mod convențional un singur astfel de fisier plasat în radacina proiectului.

După crearea a două sau mai multe branch-uri acestea sunt dezvoltate separat, conținând informații diferite deobicei. Dacă dorim să unim aceste două branch-uri într-unul singut folosim comanda *git merge*. Aceasta se va executa numai dacă între aceste branch-uri nu există conflict, adică conținutul lor este identic. Pentru a rezolva aceste conflicte trebuie sincronizate cele două branch-uri, ambele să aibă informație identică.

În istoria modificărilor pot fi marcate cele mai importante modificări cu ajutorul tag-urilor. Git-ul folosește două tipuri de tag-uri: simple și adnotate. Numele tag-urilor au următoarea formă: v0.1, v3.4 etc. Pentru a inițializa un gat adnotat introducem: *git tag -a v1.2 -m "nume-tag"*. Tag-urile adnotate sunt stocate ca obiecte complete în baza de date. Tag-ul ușor este de fapt un pointer către un anumit commit. Acesta se inițializează: textitgit tag v1.2 -lw.

4.2 Imagini

Adauga cite cel putin o imagine (sau mai multe) pentru fiecare functionalitate adaugata.

Concluzie

Aici trebuie sa fie concluzia ta.

article
test.