**Ministerul Educaţiei al Republicii Moldova**

**Universitatea Tehnică a Moldovei**

**Facultatea Calculatoare, Informaticǎ şi Microelectronicǎ**

**Departamentul Ingineria Software și Automatică**

**Raport**

**Lucrare de laborator nr. 1**

**la disciplina:** *Programarea în rețea*

***Tema: Versionarea codului sursă utilizând GIT***

**A verificat**: lector asistent Latu Eugenia

lector asistent Ostapenco Stepan

**A efectuat:** st. gr. TI-142 Chirica Alexandru

**Chișinău 2017**

**Obiectivele lucrării**

Crearea unui repozitoriu distant, localizat de serviciul **github.com**, 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:

# https://github.com/kivlara-mujik/PR1

# Introducere

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 îndepartate, sa poata lucra simultan la proiect, urmând ca, la final, modificarile lor sa 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 înlaturarea bug-urilor, iar cealaltă, în care se urmărește adăugarea de noi functionalitați, înaintea șlefuirii celor existente [1].

Există două modele de VCS-uri. Modelul centralizat (ex: SVN): codul sursa este situat pe un server central, de unde clienţii pot obţine variante de lucru pe maşina locala (**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 [2].

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;
* 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](http://en.wikipedia.org/wiki/Secure_Shell) 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](http://en.wikipedia.org/wiki/Asymmetric_key). Criptografia asimetrică este un tip de [criptografie](https://ro.wikipedia.org/wiki/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 [3].

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.

# Comenzi GIT utilizate

În cadrul acestei lucrări de laborator, pentru realizarea sarcinilor propuse, care vizează versionarea codului sursă la dezvoltarea unui proiect, au fost utilizate o serie de comenzi absolut necesare pentru realizarea scopului acestei lucrări.

Pentru a gestiona eficient dezvoltarea unui proiect în cadrul sistemului de versionare GIT, este necesar de a crea un repozitoriu distant și unul local. Pentru a inițializa un repozitoriu local GIT, într-un directoriu, se utilizează comanda **git init**. Inițial se crează un repozitoriu gol. În cadrul acestui repozitoriu se vor păstra fișierele necesare, local.

Odată ce a fost creat un repozitoriu local, utilizatorul are posibilitatea de a configura numele și adresa poștei electronice a acestuia. Această informație este inclusă în cadrul fiecărui commit realizat în cadrul dezvoltării proiectului și ulterior nu poate fi modificată. Aceste setări sunt suficient de realizat doar o singură dată, la crearea repozitoriului local. Ele vor fi utilizate de sistemul GIT pentru toate operațiunile pe care utilizatorul le realizează în cadrul acestuia. Sintaxa acestor comenzi sunt:

git config --global user.name "User Name" - pentru a seta numele utilizatorului, git config --global user.email user@gmail.com - pentru a seta poșta electronică a utilizatorului. În cazul în care, pentru dezvoltarea altui proiect este necesar de specificat alte date personale privind poșta electronică sau nume utilizatorului, se poate executa aceiași comandă fără parametrul –global, în catalogul cu proiectul necesar.

Următorul pas, este de a adăuga fișier README în repozitoriul local, prin intermediul comenzii touch README. Deși realizarea acestui pas nu este obligatorie, existența acestui fișier în cadrul proiectului este necesară, deoarece poate include o serie de informații utile, care facilitează gestionarea și dezvoltarea proiectului. Acest fișier poate include:

* informații despre cerințele sistemului;
* instrucțiuni privind instalarea programului;
* instrucțiuni privind anumite setări;
* instrucțiuni privind administrarea programului;
* informații despre licență și despre drepturi de autor;
* date de contact- pentru a contacta cu autorul proiectului;
* erori cunoscute;
* instrucțiuni privind eliminarea defectelor;
* date privind autorul;
* protocolul pentru eventuale modificări ale programului.

În cadrul repozitoriului GIT, având o copie a fișierelor de lucru pentru un anumit proiect, urmează să se realizeze anumite modificări în cadrul fișierelor acestora. Modificările vor fi fixate în așa numite snapshots a acestor modificări în repozitoriu, de fiecare dată, când proiectul ajunge la starea, la momentul căreia proiectul ar trebui fi salvat.

Un fișier în repozitoriul local poate să se afle în două stări: tracked și untracked. Fișierele aflate în starea tracked, sunt fișierele care în ultimul snapshot, ele pot fi nemodificate, modificate sau pregătite pentru commit (aflat în starea staged). Fișierele aflate în starea untracked sunt toate celelalte fișiere, oricare fișiere în repozitoriu, care nu fac parte din ultimul snapshot și nu este pregătit pentru commit. Odată cu clonarea proiectului de pe repozitoriul distant, toate fișierele se vor afla în starea tracked și nemodificate, deoarece ele sunt importate din repozitoriu distant și până când ele nu au fost redactate. Atunci când aceste fișiere vor fi redactate, GIT le va considera în calitate de fișiere modificate, deoarece au fost modificate după realizarea ultimului commit. Ciclul de viață a fișierelor în perioada dezvoltării și gestionării unui proiect sunt este reprezentat în figura de mai jos.



**Figura** 1 - File Status Lifecycle

Pentru a determina starea fișierului, se utilizează comanda de bază git status. La fel această comandă arată la moment în ce ramură se modifică fișierele.

Pentru a urmări fișierele care au fost modificate și adăugarea acestora în cadrul controlului de versionare a codului, se utilizează comanda git add. Fișierele urmărite vor fi indexate, aflându-se în secțiunea Changes to be committed. Dacă se va realiza un commit la acest moment, versiunea fișierelor proiectului care există la momentul executării comenzii git add, se va adăuga în istoria stărilor snapshots.

Odată ce fișierele modificate au fost indexate prin intermediul comenzii precedente, se pot fixa toate modificările. Metoda cea mai simplă de fixare a modificărilor în cadrul fișierelor proiectului, este de a executa comanda git commit -m “first commit”. GIT crează un commit cu mesajul spcificat. La fixarea modificărilor fișierelor în cadrul repozitoriului local, se raportează anumite informații despre denumirea ramurii în cadrul căruia s-a efectuat commit, câte fișiere au fost modificate, statistica rândurilor adăugate sau șterse în cadrul fișierelor, identificatorul unic al acestui commit. Fișierele care nu sunt indexate nu vor nimeri în snapshots. Această comandă permite crearea unui snapshot a stării proiectului la momentul dat de timp, care ulterior ar putea fi restabilit sau comparat cu versiunea actuală a proiectului.

Pentru a avea posibilitatea de a lucra cu un proiect GIT, se utilizează repozitorii distante. Un repozitoriu distant se păstrează în rețea. Ele pot fi mai multe, fiecare dintre ele de regulă, ar trebui să fie disponibil sau doar pentru citire, sau pentru citire și înscriere. Pentru a obține accesul la un repozitoriu distant, se utilizează comanda git remote add origin [URL].

Pentru a distribui fișierele în repozitoriul distant, se utilizează comanda git push -u [serverul distant][ramura]. Această comandă se va executa numai în cazul în care accesul la repozitoriul distant nu este restricționat, și utilizatorul are drepturi de a efectua modificări în repozitoriu.

Aceste ramuri pot fi într-un final unite, adică modificările efectuate, să fie adăugate în ramura de bază. Aceasta se efectuează prin intermediul comenzii git merge NewBranch. La executarea acestei comenzi ar putea apărea eventuale conflicte, care specifică că aceeași secvență de cod sunt modificate diferit în cadrul celor două ramuri. Odată cu soluționarea acestor conflicte, cu succes se poate crea un snapshot, efectuând un commit [4].

**Limbajul de programare Python**

Python este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez Guido van Rossum. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C.

Python este un limbaj multifuncțional folosit de exemplu de către companii ca Google sau Yahoo! pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python. Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați și chiar și la predarea limbajului în unele medii universitare. Din aceleași motive, multe sisteme bazate pe Unix, inclusiv Linux, BSD și Mac OS X includ din start interpretatorul CPython.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C.

În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul object-oriented, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu „gunoier” (garbage collector). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode [5].

**Dependența de biblioteci externe**

Controlul de existență a unei biblioteci externe a fost realziat cu ajutorul constructiei **try** și **except**. În corpul instrucțiunii **try** se include biblioteca externă, în caz dacă biblioteca externa nu există programul se finisează cu eroare și descrierea instrucțiunii de instalare a bibliotecii externe. Aceasta este o metodă de control de dependențe implementată în interiorul codului. Programul nu va lucra pînă dependențele necesare nu o să fie instalate.

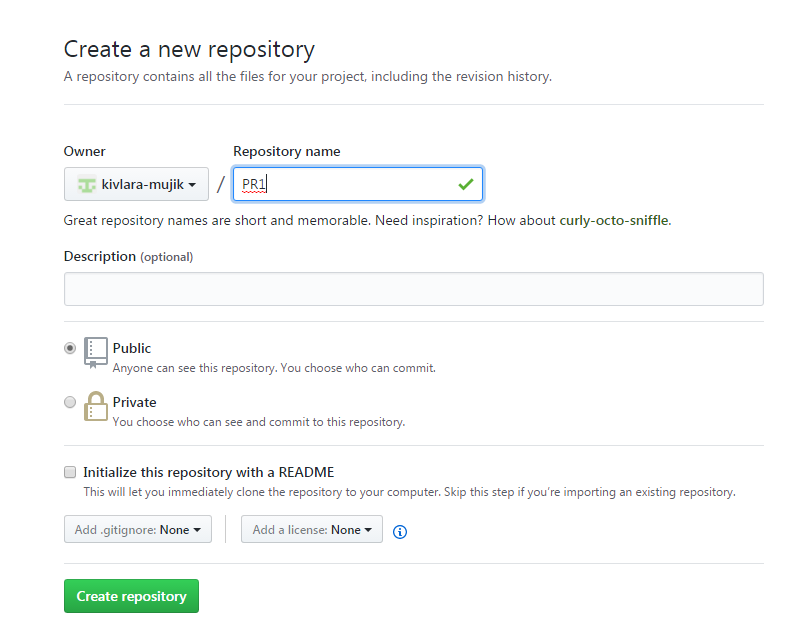
Biblioteca necesară BeautifulSoup este clonată de pe github.com și instalată prin intermediul limbajului de programare Python.

**Mersul lucrării**

Adresa repozitorului: https://github.com/kivlara-mujik/PR1/tree/master/Lab-1

Commiturile: https://github.com/kivlara-mujik/PR1/commits/master

##### 1. Crearea repozitoriului pe platforma *github.com*

**Figura** 2 - Pagina de creare a repositoriului.

##### 2. Configurarea SSH key

Acesînd pagina ***github.com/settings/keys*** adăugăm public key-ul calculatorului nostru. Public Key-ul îl primim în terminal.

Generarea public-key:

**$** *ssh-keygen*

Afișarea public-key:

**$** *cat ~/.ssh/id\_rsa.pub*

Rezultatul obținut îl salvăm în forma de adăugare a unui nou SSH key.

##### 3. Inițializarea repozitoriului GIT în directoriul local de lucru.

Clonăm și totodată initializăm repozitoriul într-un nou directoriu.

**$** git clone *https://github.com/kivlara-mujik/PR.git*

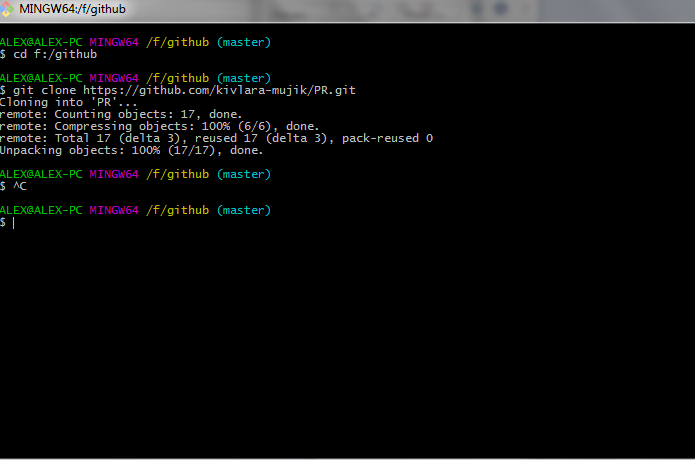


Figura 3 – Fereastra GitHub cu clonarea repositoriului PR

##### 4. Configurarea utilizatorului local

Numele de utilizator trebuie să coincidă cu acel din github, în caz contrar commit-urile se vor îndeplini din partea unui utilizator neindentificat.

*$ git config --global user.name "kivlara-mujik"*

*$ git config --global user.email "chiricaxander@gmail.com"*

##### 5. Adăugăm fisierul README

**$** *vim README.md*

După ce am modificat continutul fisierului îl încarcăm în repozitoriu.

$ *git add -A*

$ *git commit -m “Adaugarea fisierului README.md”*

$ *git push*

##### 6. Crearea directoriei laboratorului “1”

**$** *mkdir Lab-1*

**$** *cd Lab-1*

**$** *vim README.md*

**$** *git add -A*

**$** *git commit -m “Laboratorul 1”*

**$** *git push*

##### 7. Crearea a două fișiere Python

1. **$** *cd Lab-1*
2. **$** *touch helpers.py*
3. **$** *touch main.py*
4. **$** *git add -A*
5. **$** *git commit -m “Fisierele initiale”*
6. **$** *git push*

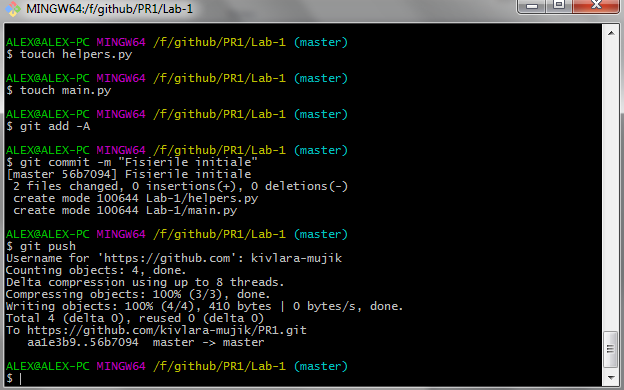
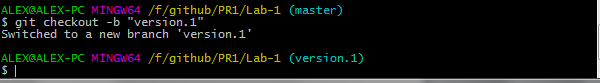


Figura 4 – Crearea a doua fisiere Python

##### 8. Crearea și trecerea la altă ramură

**$** *git checkout -b “version.1”*



**Figura 5** – Crearea unui branch nou

##### 9. Adăugarea unui fișier în ramura nouă

**$** *touch setari.py*

**$** *git add -A*

**$** *git commit -m “Fisierul setărilor”*

**$** *git push origin version.1*

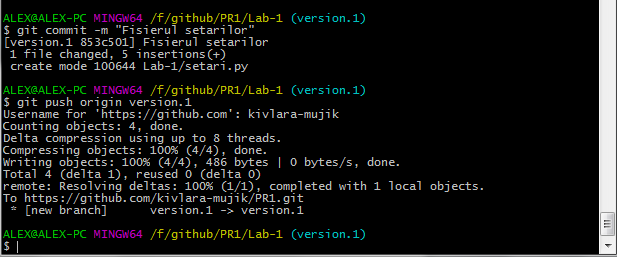
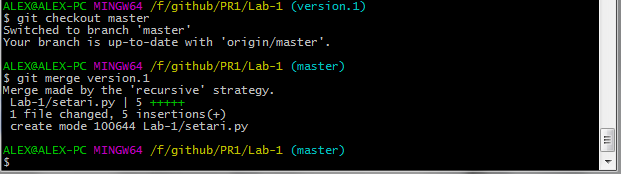


Figura 6 – Adaugarea fisierului in noul branch

##### 10. Merging

$ git checkout master

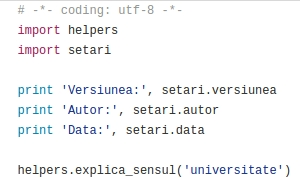
$ git merge version.1



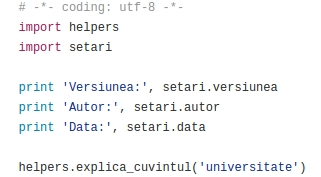
**Figura 7** – Facem merge la version.1

**11. Crearea unui conflict între commit-uri**

$ *git checkout -b “versiunea.2”*



**Figura** 8 - Conținutul fișierului **main.py** în ramura **versiunea.2**



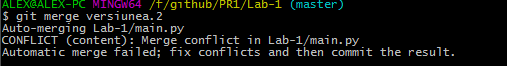
**Figura 9** - Conținutul fișierului **main.py** în ramura **master**

Diferenta e că în **master** deja au fost încărcate commit-uri cu noua denumire pentru funcția “**explica**” în “**explica\_cuvintul**”. În ramura “**versiunea.2**” denumirea funcției a fost modificată în “**explica\_sensul**”.

$ git checkout master

$ git merge versiunea.2

$ git pull



**Figura 10** – Conflictul intre commituri

**12. Rezolvarea conflictului în consolă**

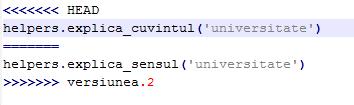
*Auto-merging* ***Lab-1/main.py***

*CONFLICT (content): Merge conflict in* ***Lab-1/main.py***

*Auto-merging* ***Lab-1/helpers.py***

*CONFLICT (content): Merge conflict in* ***Lab-1/helpers.py***

*Automatic merge failed; fix conflicts and then commit the result.*



**Figura 11** - Modificarile făcute automat în urma depistării conflictului

Modificăm manual continutul fisierelor main.py și helpers.py lăsînd denumirea functiei din ramura versiunea.2 “**explica\_sensul**”. Dupa îndeplinirea modificarilor încarcăm schimbările în ramura “**master**”.

**$** *git add -A*

**$** *git commit -m “Rezolvarea conflictului”*

**$** *git push origin master*

**13. Dependența de biblioteci externe**

Realizarea dependenței de biblioteci externe a fost realziata direct în cod. În momentul rulării programului sunt incluse biblitoecile necesare, în caz de eroare (ca urmare lipsa bibliotecii externe) programul afișează mesajul de înlaturare a greșelii prin instalarea bibliotecii necesare.



**Figura 12** – Verificarea existenței a unei biblioteci externe

**Concluzii**

Sistemul de versionare GIT este un mod eficient de management al fișierelor care permite păstrarea istoricului tuturor modificărilor aduse fișierelor urmărite. Unul dintre avantajele folosirii sistemului GIT este posibilitatea de a reveni la o versiune mai veche a proiectului. Utilizatorul dispune de o copie de siguranță a codului sursă. Cea mai recentă versiune a codului sursă este mereu disponibilă tuturor dezvoltatorilor. GIT simplifică procesul de distribuire a codului sursă și să înlesnește colaborarea pe proiecte. Un moment mai dificil pe parscursul elaborarii laboratorului a fost rezolvarea de conflicte între două ramuri.

Limbajul de programare ales este Python. Acest limbaj a fost ales din cauza existenței a cunoștințelor necesare și dorința de a practica acest limbaj în mai multe domenii. Limbajul este intuitiv clar. În laboratorul curent am realziat verificarea dependențelor de alte biblioteci direct în cod. Pentru Python există un manager de biblioteci numit pip. El automatizează verificările dependenților și îndeplinește update pentru bibliotecile care au deja o versiune nouă.

Am analizat structura HTML unei pagini. În aplicație am elaborat o funcție care caută explicație a unui cuvînt înntr-un dicționar explicativ român. Biblioteca BeautifulSoup parsează conținutul paginii și a returnat elementele necesare mie în dependență de unii parametri.

În unele din commit-uri am folosit Regular Expresion pentru a parsa continutul HTML a paginii. Dar după apariția unor dificultăți cu adaptarea conținutului HTML sub un anumit standart am decis să folosesc o biblitoecă externă care a ușurat lucrul. În acest moment o concluzie importantă la care am ajuns este faptul că în GitHub deja sunt sute de biblioteci gata care rezolvă o mare parte a problemelor apărute

**Bibliografie**

1. Scott Chacon, Pro Git [Resursă electronică]. 23 februarie 2017

<http://git-scm.com/book>

2. Lars Vogel, Git - Tutorial [Resursă electronică]. 10 august 2015

<http://www.vogella.com/tutorials/Git/article.html>

3. Atlassian, Git Tutorials [Resursă electronică]. 23 februarie 2017

<https://www.atlassian.com/git/tutorial>

4. Vincent Driessen, A successful Git branching model [Resursă electronică].

<http://nvie.com/posts/a-successful-git-branching-model/> 5 ianuarie 2010

5. Introducere în Python [Resursă electronică]. 13 iulie 2009

<http://www.e-learn.ro/tutorial/python/introducere-in-python-partea-i/149/1/367.htm>