Inginerie Software 2024
Laboratorul 9
Versionarea Software

Mădălina Erașcu, Alexandru Munteanu, Cristian Pal, Ionica Puiu

April 2024

1 Versionarea software

În domeniul ingineriei software [2], *versionarea software* se referă la practica de a gestiona și controla versiunile diferitelor artefacte software (cum ar fi codul sursă, documentația, resursele) pe măsură ce acestea sunt dezvoltate și schimbate de-a lungul timpului. Scopul principal al versionării este de a urmări modificările făcute asupra codului și de a gestiona aceste modificări într-un mod controlat, organizat și colaborativ.

Iată câteva aspecte cheie ale versionării software:

1.1 Controlul versiunilor

Prin utilizarea unui sistem de control al versiunilor[1] (VCS - Version Control System), dezvoltatorii pot urmări modificările, adăugirile și ștergerile de cod într-un proiect de-a lungul timpului. Acest lucru permite revenirea la versiuni anterioare, compararea diferențelor între versiuni și gestionarea modificărilor de către mai mulți dezvoltatori într-un mod organizat.

1.2 Istoricul modificărilor

O componentă esențială a versionării este să păstrezi un istoric al tuturor modificărilor făcute asupra codului. Acesta oferă o înregistrare clară a cine a făcut ce modificare, când și de ce, ceea ce este util pentru depanare, revizuirea codului și înțelegerea evoluției proiectului.

1.3 Colaborare eficientă

Când mai mulți dezvoltatori lucrează la același proiect, un sistem de control al versiunilor permite colaborarea fără probleme. Dezvoltatorii pot lucra pe ramuri separate (branches) ale codului, aducându-și contribuțiile individuale, iar apoi integrându-le în ramura principală (branch-ul principal sau master) când sunt gata.

1.4 Gestionarea conflictelor

În timpul colaborării, este posibil ca doi sau mai mulți dezvoltatori să facă modificări în același fișier sau secțiune de cod. Sistemele de control al versiunilor facilitează identificarea și rezolvarea acestor conflicte pentru a menține integritatea și funcționalitatea codului.

1.5 Livrarea versiunilor

Odată ce o anumită versiune a software-ului este pregătită pentru lansare, sistemul de control al versiunilor poate ajuta la etichetarea acelei versiuni specifice (tagging) pentru a păstra o referință clară a ceea ce a fost livrat.

Sistemele de control al versiunilor populare includ Git[1], SVN (Subversion)[2], Mercurial[3], etc. Acestea sunt instrumente esențiale în dezvoltarea modernă a software-ului și ajută la organizarea, colaborarea și gestionarea evoluției proiectelor de software.

2 Managementul Versiunilor

Managementul versiunilor în dezvoltarea software implică gestionarea eficientă a diferitelor versiuni ale produsului, de la versiuni de bază până la patch-uri și versiuni de întreținere. Release-urile sunt etape importante în lansarea și distribuția software-ului, iar branch-urile de development și feature permit dezvoltatorilor să lucreze în mod colaborativ și organizat la îmbunătățirea produsului. Este esențial să se utilizeze un sistem de control al versiunilor pentru a urmări și gestiona aceste versiuni și modificări într-un mod controlat și eficient.

- Versiuni de Bază este prima versiune a unui produs software, adesea denumită și "v1.0" sau "1.0".Reprezintă versiunea inițială, lansată pentru utilizare generală.
- Versiune Patch este o versiune minoră a software-ului care adresează probleme specifice sau vulnerabilități.Este denumită adesea "v1.0.1" sau "1.0.1", indicând că este o actualizare mică a versiunii de bază.
- Versiune MR (Maintained Release) este o versiune a software-ului care continuă să primească suport și actualizări de la dezvoltatori.Este importantă pentru clienții care doresc să aibă o versiune stabilă și sigură, cu suport pe termen lung.
- Release-uri sunt versiunile de software care sunt pregătite pentru distribuție și lansare publică. Pot include noi caracteristici, îmbunătățiri, patch-uri de

[1]Pentru mai multe informații despre Git, consultați documentația oficială disponibilă la `https://git-scm.com/doc`.

[2]Pentru mai multe informații despre SVN (Subversion), consultați documentația oficială disponibilă la `https://subversion.apache.org/docs/`.

[3]Pentru mai multe informații despre Mercurial, consultați documentația oficială disponibilă la `https://www.mercurial-scm.org/guide`.

securitate și alte actualizări. Sunt etichetate și marcate pentru a diferenția între diferitele versiuni de software.

- Branch de Development este o ramură (branch) a codului care este folosită pentru dezvoltarea continuă a produsului. Dezvoltatorii lucrează pe această ramură pentru a adăuga funcționalități noi și pentru a face modificări. După ce funcționalitățile sunt testate și sunt gata pentru lansare, acestea sunt încorporate în branch-ul principal.

- Branch-uri Feature sunt ramuri separate create pentru a lucra la anumite funcționalități noi sau la modificări importante. Dezvoltatorii lucrează pe aceste ramuri pentru a evita interferența cu codul principal și pentru a facilita colaborarea. După finalizarea unei funcționalități, branch-ul de feature este încorporat în branch-ul de development sau principal.

3 Repository

La baza sistemului de control al versiunilor se află un depozit, care este depozitul central al datelor sistemului respectiv. De obicei, depozitul stochează informații sub forma unui arbore de sistem de fișiere care este compus la rândul ei dintr-o ierarhie de fișiere și directoare. Orice număr de clienți care se conectează la acest depozit, au posibilitatea de a citi sau scrie în ierarhia de fișiere și directoare. Prin scrierea datelor, un client pune informațiile la dispoziția altora iar prin citirea datelor clientul primește informații de la alții.

Până în acest punct un repository nu este nimic altceva decât un server de fișiere tipic, dar ce îl face cu adevărat special este că, pe măsură ce fișierele din repository sunt modificate, repository-ul poate aduce orice versiune a acelor fișiere. Când un client citește date din depozit, în mod normal vede doar cea mai recentă versiune a arborelui sistemului de fișiere. Un client de asemenea poate cere un istoric despre un fișier particular precum "Ce a conținut acest director săptămâna trecută?" și "Cine a fost ultima persoană să schimbe acest fișier și ce modificări a făcut?".Acestea sunt tipurile de întrebări care se află în centrul oricărui sistem de control al versiunilor. De exemplu pentru a afla ce a conținut un director săptămâna trecută, putem folosi următoarea comandă Git:

```
git log --since="1 week ago" --pretty=format:"%h - %an, %ar :  %s"
```

Pentru a afla cine a fost persoana care a făcut ultima modificare asupra unui fisier putem folosi urmatoarea comanda Git:

```
git blame nume_fișier
```

Aceasta abordare este ilustrată în Figura 1 de mai jos.

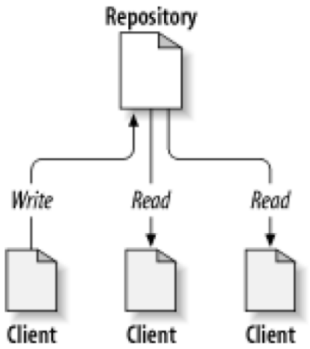

Figure 1: Exemplu repository

4 Copia de lucru

Valoarea unui sistem de control al versiunilor provine din faptul că urmărește versiunile de fișiere și directoare, dar restul programelor software (ex. IDE-ul) nu operează pe "versiuni de fișiere și directoare".Aceste programe știu să opereze doar pe o singura versiune al unui fișier.Așadar răspunsul la această problemă se găsește în constructul de control al versiunii cunoscut sub numele de copie de lucru.

O copie de lucru este, la propriu, o copie locală a unei anumite versiuni a datelor/aplicației gestionate de VCS ale unui utilizator pe care utilizatorul este liber să facă modificări. Copia de lucru apare ca fișiere și directoare locale într-un program de editare spre exemplu, iar aceste programe nu trebuie sa fie "Version-control-aware" pentru a citi și a scrie în acele fișiere sau directoare. Sarcina de a gestiona copia de lucru și de a comunica modificările aduse conținutului său către și de la depozit revin direct la software-ul client al sistemului de control al versiunilor.Acest concept este ilustrat în Figura 2.

5 Git

5.1 Introducere

Git[3] este un sistem version control care rulează pe majoritatea platformelor, inclusiv Linux, POSIX, Windows și OS X. Ca și Mercurial, Git este un sistem distribuit și nu întreține o bază de date comună. Este folosit în echipe de dezvoltare mari, în care membrii echipei acționează oarecum independent și sunt răspândiți pe o arie geografică mare.Git este dezvoltat și întreținut de Junio Hamano, fiind publicat sub licență GPL și este considerat software liber. Dintre proiectele majore care folosesc Git amintim Amarok, Android, Arch Linux, Btrfs, Debian, DragonFly BSD, Eclipse, Fedora, FFmpeg , GIMP, GNOME, GTK+, Hurd, Linux kernel, Linux Mint, openSUSE, Perl, phpBB, Qt, rsync,

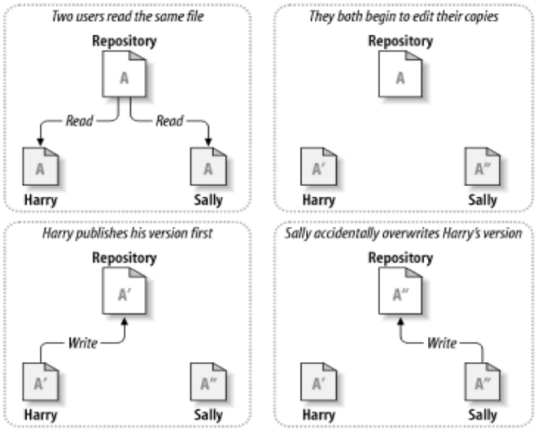

Figure 2: Exemplu pentru copia de lucru

Ruby on Rails, Samba.

Ca și o scurtă istorie Git a luat naștere atunci când relația dintre BitKeeper si Linux Kernel s-a rupt. Să ne reamintim dacă tot a venit vorba ce este Linux Kernel – este un proiect software cu sursă deschisă de o amploare destul de mare. În primii ani de întreținere a nucleului Linux (1991–2002), modificările aduse software-ului au fost transmise ca patch-uri și fișiere arhivate. În 2002, proiectul Linux kernel a început să folosească un DVCS proprietar numit BitKeeper.

În 2005, după cum am amintit relația dintre comunitatea care a dezvoltat nucleul Linux și compania comercială care a dezvoltat BitKeeper s-a rupt, iar statutul gratuit al instrumentului a fost revocat. Acest lucru a determinat comunitatea de dezvoltare Linux (și în special Linus Torvalds, creatorul Linux) să-și dezvolte propriul instrument bazat pe unele dintre lecțiile pe care le-au învățat în timpul utilizării BitKeeper. Unele dintre obiectivele noului sistem au fost următoarele:

- Viteză
- Design simplu
- Suport puternic pentru dezvoltarea neliniară (mii de ramuri paralele)
- Distribuit complet
- Capabil să gestioneze proiecte mari, cum ar fi kernel-ul Linux, în mod eficient (viteză și dimensiunea datelor)

De la nașterea sa în 2005, Git a evoluat și s-a maturizat pentru a fi ușor de utilizat și, totuși, își păstrează aceste calități inițiale. Este uimitor de rapid, este foarte eficient pentru proiecte mari și are un sistem de ramificare incredibil pentru dezvoltare neliniară.

5.2 Repository în Git

Un depozit Git este pur și simplu o bază de date care conține toate informațiile necesare pentru a păstra și gestiona revizuirile și istoricul unui proiect. În Git, ca și în cazul majorității sistemelor de control al versiunilor, un depozit păstrează o copie completă a întregului proiect pe toată durata de viață. Cu toate acestea, spre deosebire de majoritatea celorlalte VCS, depozitul Git oferă nu numai o copie completă de lucru a tuturor fișierelor din depozit, ci și o copie a depozitului însuși cu care să lucreze.

Git menține un set de valori de configurare în fiecare depozit cum ar fi numele și adresa de e-mail a utilizatorului depozitului. Spre deosebire de datele fișierului și alte metadate ale depozitului, setările de configurare nu sunt propagate de la un depozit la altul în timpul unei operațiuni de clonare sau duplicare. În schimb, Git gestionează și inspectează informațiile de configurare pe site, pe utilizator și pe depozit. Un depozit Git este fie un depozit simplu, fie un depozit de dezvoltare (nonbare). Un depozit de dezvoltare este folosit pentru dezvoltarea normală, zilnică. Menține noțiunea de ramură curentă și oferă o copie extrasă a ramurii curente într-un director de lucru. În schimb, un depozit simplu nu are un director de lucru și nu ar trebui să fie folosit pentru dezvoltare normală.

Cu alte cuvinte, nu ar trebui să facem comit-uri directe către un depozit complet. Un depozit simplu poate părea de puțin folos, dar rolul său este important pentru a servi ca baza autoritară pentru dezvoltarea colaborativă.

Alți dezvoltatori clonează și preiau din depozitul complet și introduc actualizări către acesta. Dacă emiteți git clone cu opțiunea –bare, Git creează un depozit complet; în caz contrar, se creează un depozit de dezvoltare. În mod implicit, Git activează un reflog (o înregistrare a modificărilor aduse referințelor) în depozitele de dezvoltare, dar nu și în depozitele necompletate. Acest lucru anticipează din nou că dezvoltarea va avea loc în primul și nu în cel din urmă. Prin același raționament, nici un remote repository nu este creată într-un mod liber.

Dacă configurăm un depozit în care dezvoltatorii introduc modificări, acesta ar trebui să fie complet. De fapt, acesta este un caz special al celei mai bune practici mai generale conform căreia un depozit publicat ar trebui să fie complet.

În cadrul unui depozit, Git menține două structuri de date primare, depozitul de obiecte și indexul. Toate aceste date din depozit sunt stocate la rădăcina directorului de lucru într-un subdirector ascuns numit .git. Colecția de obiecte este conceput pentru a fi copiat eficient în timpul unei operațiuni de clonare, ca parte a mecanismului care acceptă un VCS complet distribuit. Indexul reprezintă informații tranzitorii, este de asemenea privat pentru un depozit și poate fi creat sau modificat la cerere, după cum este necesar.

5.3 Colecție de obiecte Git

În centrul implementării depozitului Git este depozitul de obiecte. Conține fișierele de date originale și toate mesajele de log, informațiile despre autor, datele și alte informații necesare pentru a reconstrui orice versiune sau pentru

a reconstrui orice ramură a proiectului și respectiv pentru a reconstrui orice etichetă. Aceste patru obiecte atomice formează baza datelor de nivel superior Git structuri:

- Blobs — Fiecare versiune a unui fișier este reprezentată ca un blob. "Blob" este o contracție a unui "obiect binar mare", un termen care este folosit în mod obișnuit în calcul pentru a se referi la o variabilă sau un fișier care poate conține orice date și a cărui structură internă este ignorată de program. Un blob este tratat transparent. Un blob deține datele unui fișier, dar nu conține metadate despre fișier sau chiar numele acestuia.

- Trees — Un obiect arbore reprezintă un nivel de informații despre director. Înregistrează identificatorii blob, căile și o fracțiune de metadate pentru toate fișierele dintr-un singur director. De asemenea, poate face referire recursiv la alte (sub)obiecte din arbore și astfel construiește o ierarhie completă de fișiere și subdirectoare.

- Commits — Un obiect de comitere deține metadate pentru fiecare modificare introdusă în depozit, inclusiv numele autorului, data de comitere și mesajul de jurnal. Fiecare commit indică un obiect arborescent care surprinde, într-un instantaneu complet, starea depozitului la momentul în care a fost efectuată commit-ul. Comitarea inițială, sau comiterea rădăcină, nu are părinte. Majoritatea comiterilor au un singur părinte de comitere.

- Etichete — Un obiect etichetă atribuie un nume arbitrar; un exemplu de astfel de etichetă este `9da581d910c9c4ac93557ca4859e767f5caf5169` care se referă la un commit exact și bine definit, un nume de etichetă mai familiar ar putea avea mai mult sens!

De-a lungul timpului, toate informațiile din depozitul de obiecte se schimbă și cresc, urmărind și modelând editările, completările și ștergerile proiectului. Pentru a utiliza eficient spațiul pe disc și lățimea de bandă a rețelei, Git comprimă și stochează obiectele în fișiere pachet, care sunt, de asemenea, plasate în depozitul de obiecte.

5.4 Indexul

Indexul este un fișier binar temporar și dinamic care descrie structura de directoare a întregului depozit. Mai precis, indexul surprinde o versiune a structurii generale a proiectului la un moment dat. Starea proiectului poate fi reprezentată de un commit și un arbore din orice punct din istoria proiectului sau poate fi o stare viitoare a proiectului. Una dintre caracteristicile distinctive cheie ale Git este că ne permite să modificăm conținutul indexului în pași metodici, bine definiți. Indexul permite o separare între etapele de dezvoltare incrementale și efectuarea acelor modificări. În calitate de dezvoltator, se execută comenzi Git pentru a introduce modificări în index. Modificările de obicei adaugă, șterg sau editează un fișier sau un set de fișiere. Indexul înregistrează și păstrează

acele modificări, păstrându-se în siguranță până când le vom folosi.De asemenea, putem elimina sau înlocui modificările din index. Astfel, indexul permite o tranziție treptată, de obicei ghidată., de la o stare complexă din depozit la alta, probabil mai bună. Indexul joacă un rol important în îmbinări, permițând versiuni multiple ale aceluiași fișierul care urmează să fie gestionat, inspectat și manipulat simultan.

5.5 Branch și Merge în Git

5.5.1 Branch (Ramură)

O ramură este mijlocul fundamental de lansare a unei linii separate de dezvoltare în cadrul unui proiect software. O ramură este o separare dintr-un fel de stare primară unificată, care permite dezvoltării să continue în mai multe direcții simultan și, potențial, să producă versiuni diferite ale proiectului. Adesea, o ramură este reconciliată și fuzionată cu alte ramuri pentru a reuni eforturile. Git permite multe ramuri și, prin urmare, multe linii diferite de dezvoltare în cadrul unui depozit. Sistemul de ramificare al lui Git este ușor și simplu. Mai mult, Git are suport de prim rang pentru îmbinări (merge). Ca rezultat, majoritatea utilizatorilor Git folosesc de rutină acest feature de branching.

5.5.2 Motive pentru utilizarea ramurilor

O ramură poate fi creată din nenumărate motive tehnice, filozofice, manageriale și chiar sociale. În continuare o sa prezentăm câteva argumente comune:

- O ramură reprezintă adesea o versiune individuală a clientului. Dacă dorim să începem versiunea 1.1 a proiectului, dar știm că unii dintre clienții noștri. doresc să rămână cu versiunea 1.0, păstrăm versiunea veche în viață ca o ramură separată.
- O ramură poate incapsula o fază de dezvoltare, cum ar fi lansarea prototipului, beta, stabilă sau de ultimă oră. Ne putem gândi și la versiunea "1.1" ca o fază separată - versiunea de întreținere.
- O ramură poate izola dezvoltarea unei singure caracteristici sau cercetarea unui bug deosebit de complex. De exemplu, putem introduce o ramură pentru o sarcină bine definită și izolată conceptual sau pentru a facilita o îmbinare a mai multor ramuri înainte de o lansare.Poate pare exagerat crearea unei astfel crearea unei astfel de ramuri dar Git încurajează o astfel de utilizare la scară mică.
- O ramură individuală poate reprezenta munca unui colaborator individual. O altă ramură – ramura de "integrare" – poate fi folosită în mod special pentru unificarea eforturilor.

Git se referă la o ramură, cum ar fi cele enumerate doar ca o ramură de subiect sau o ramură de dezvoltare. Cuvântul "subiect" indică pur și simplu că fiecare ramură din depozit are un anumit scop.

5.5.3 Merge în Git

Git este un sistem distribuit de control al versiunilor (DVCS). Permite unui dezvoltator din România, de exemplu, și altuia din New York să facă și să înregistreze modificări în mod independent și le permite celor doi dezvoltatori să-și combine modificările în orice moment, toate fără un depozit central. O îmbinare (sau merge) unifică două sau mai multe ramuri ale istoricului de comitere. Cel mai adesea, o îmbinare unește doar două ramuri, deși Git acceptă o îmbinare a trei, patru sau mai multe ramuri în același timp. În Git, o îmbinare trebuie să aibă loc într-un singur depozit - adică toate ramurile care trebuie îmbinate trebuie să fie prezente în același depozit. Modul în care ramurile ajung să fie în depozit nu este important. Când modificările dintr-o ramură nu intră în conflict cu modificările găsite într-o altă ramură, Git calculează un rezultat de îmbinare și creează un nou commit care reprezintă starea nouă, unificată. Dar atunci când ramurile sunt în conflict, care apare ori de câte ori schimbările concurează pentru a modifica aceeași linie a aceluiași fișier, Git nu rezolvă disputa (această problemă este rezolvată de către un developer). În schimb, Git marchează în index astfel de modificări controversate ca "neunificate" și vă lasă reconcilierea în seama, dezvoltatorului. Când Git nu poate îmbina automat, depinde și de comitter să se efectueze comitarea finală odată ce toate conflictele sunt rezolvate. Înainte de a începe o îmbinare, este recomandat verificarea directorului de lucru.În timpul unei îmbinări obișnuite, Git creează versiuni noi de fișiere și le plasează în directorul curent de lucru. Mai mult, Git folosește și indexul pentru a stoca versiuni temporare și intermediare ale fișierelor în timpul operațiunii. Dacă am modificat fișiere în directorul de lucru sau dacă am modificat indexul prin git add sau git rm, atunci depozitul are un director de lucru murdar (dirty). Dacă începem o îmbinare într-o stare murdară, este posibil ca Git să nu poată combina modificările din toate ramurile și cele din directorul sau indexul de lucru într-o singură trecere.

5.6 Serviciul GitHub

Github deseori este confundat cu Git în sine dar ele cu adevărat sunt diferite... Git este un sistem distribuit de control al versiunilor pentru urmărirea modificărilor codului sursă în timpul dezvoltării software. Este conceput pentru coordonarea muncii între programatori, dar poate fi folosit pentru a urmări modificările în orice set de fișiere. Obiectivele sale includ viteza, integritatea datelor și suport pentru fluxuri de lucru distribuite, neliniare iar GitHub reprezintă un serviciu de găzduire a depozitelor Git bazat pe web, care oferă toate funcționalitățile distribuite de control al reviziilor și de gestionare a codului sursă (SCM) ale Git, precum și adăugarea propriilor caracteristici. Tabelul 1 prezintă diferențele majore între Git și Github:

Git	**GitHub**
Git este un Software	GitHub este un serviciu
Git este un instrument de linie de comandă	GitHub este o interfață grafică de utilizator
Git este instalat local pe sistem	GitHub este găzduit pe web
Git este întreținut de Linux	GitHub este întreținut de Microsoft
Git se concentrează pe controlul versiunilor și pe partajarea codului	GitHub se concentrează pe găzduirea centralizată a codului sursă
Git este un sistem de control al versiunilor pentru a gestiona istoricul codului sursă	GitHub este un serviciu de găzduire pentru depozitele Git
Git a fost lansat pentru prima dată în 2005	GitHub a fost lansat în 2008
Git nu are nicio funcție de gestionare a utilizatorilor	GitHub are o funcție încorporată de gestionare a utilizatorilor
Git are licență open-source	GitHub include un nivel gratuit și un nivel cu plată pentru utilizare
Git are o configurație minimă a instrumentelor externe	GitHub are o piață activă pentru integrarea instrumentelor
Git oferă o interfață desktop numită Git Gui	GitHub oferă o interfață desktop numită GitHub Desktop
Git concurează cu CVS, Azure DevOps Server, Subversion, Mercurial etc.	GitHub concurează cu GitLab, Git Bucket, AWS Code Commit etc.

Table 1: Git vs GitHub

6 Teme

1. Creeati un repository nou pe GitHub. Asociați un nume și o descriere corespunzătoare. Puteți alege că repository-ul să fie public sau privat. Încărcarea inițială vă fi compusă din temele realizate de pană acum, iar pentru următoarele teme veți utiliza acest repository pentru a face commit săptămânal pentru temele viitoare.

2. Folosind comanda `git clone`, clonați repository-ul creat anterior. Asigurați-vă că este instalat Git.

3. Creați un fișier nou sau modificați unul existent în directorul local al repository-ului clonat. Folosiți `git add` pentru a adăuga modificările în zona de staging. Apoi utilizați `git commit` pentru a confirma modificările împreună cu un mesaj descriptiv.

4. Utilizați `git push` pentru a trimite modificările făcute pe branch-ul local pe GitHub. Verificați pe platforma GitHub pentru a vă asigura că modificările au fost încărcate cu succes.

5. Creați un branch nou folosind `git checkout -b nume_branch`. Adaugați modificările în acest branch și executați un commit. Încarcați branch-ul pe GitHub folosind `git push origin nume_branch`. Apoi faceți un

pull request pentru a integra modificările din acest branch în branch-ul principal (de obicei master sau main).

References

[1] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, 2012.

[2] Ian Sommerville. *Software Engineering*. 10th ed. Pearson, 2018.

[3] Mariot Tsitoara. *Beginning Git and GitHub: A Comprehensive Guide to Version Control, Project Management, and Teamwork for the New Developer*. Apress, 2019.