Controlul versiunilor, Git

Git este un sistem de management al proiectelor software și de gestionare a versiunilor codului sursă. Un proiect software în Git se numește *repository* și poate conține fișiere cod sursă, baze de date, fișiere text cu documentație etc.
În Git fiecare versiune a unui program se numește *commit* și conține modificările aduse unuia sau mai multor fișiere. Astfel, istoricul modificărilor este conținut în lista commit-urilor și se poate reveni cu ușurință la o versiune anterioară a programelor. De aceea este important ca fiecare commit să se refere la o singură modificare logică (inclusiv în cazul în care afectează mai multe fișiere).

Datorită faptului că este un sistem distribuit, Git se pretează pentru lucrul în echipă, fiecare membru având o copie locală a software-ului, iar modificările aduse de fiecare fiind integrate pe un server.
Fiecare programator își poate alege modalitatea în care salvează aceste modificări, însă atunci când se lucrează în echipă trebuie ținut cont de faptul că ele pot afecta munca altor colegi. De aceea, este o idee bună ca versiunile (commit-urile) să fie funcționale, chiar dacă nu sunt finalizate.

Git permite de asemenea crearea unor ramuri de dezvoltare separate (denumite *branch*-uri), astfel încât funcționalitățile (eventual incomplete) create pe o anumită ramură să nu afecteze dezvoltarea întregului software. Ramura principală se numește, de regulă, `main` (sau `master`) și conține o versiune funcțională și stabilă a software-ului. Operația prin care o ramură (ajunsă în stadiul final de dezvoltare) se integrează cu o alta se numește *merge*.

Pentru început vom vedea cum se crează și cum se folosește un repository local, iar apoi cum se sincronizează acesta cu unul aflat pe server (*remote*).

1 Instalare

Dacă lucrați pe calculatorul personal, este necesar ca întâi să instalați Git, varianta pentru sistemul vostru de operare, de aici. Dacă lucrați pe un calculator din laborator, săriți acest pas, Git este deja instalat.

În timpul instalării, în fereastra **Adjusting the name of the initial branch in new repositories**, alegeți **Override the default branch name for new repositories**, cu textul **main**. Pentru restul ferestrelor puteți lăsa opțiunile default.

2 Repository local

Pentru a utiliza Git într-un proiect software e nevoie de inițializarea unui repository în directorul care conține programele din proiect. Apoi, ori de câte ori se aduc modificări unuia sau mai multor fișiere, aceste modificări vor fi incluse într-un commit nou.

Toate aceste operații se realizează în linia de comandă (Command Prompt în Windows sau Terminal în Linux).

2.1 Configurare

Pentru a inițializa un repository nou, în linia de comandă navigați în directorul dorit și folosiți comanda `git init`:

```
C:\firstrepo> git init
```

Configurați apoi user-ul, cu nume și adresa de email (pot fi fictive dacă lucrați pe un calculator din laborator):

```
C:\firstrepo> git config user.name "Codin Codescu"
C:\firstrepo> git config user.email "codin@gmail.com"
```

2.2 Utilizare

Crearea unui commit se realizează în două faze. Prima se numește *staging* și presupune semnalarea fișierelor modificate care urmează să fie integrate în noul commit. Aceasta se face utilizând comanda `git add`.

```
C:\firstrepo> git add fisier1.c fisier2.cpp
```

Această etapă reprezintă pregătirea pentru commit, iar modificările aduse aici sunt reversibile. Spre exemplu, comanda de mai jos elimină `fisier2.cpp` din lista celor ce vor fi incluse în commit:

```
C:\firstrepo> git restore --staged  fisier2.cpp
```

Puteți vedea oricând ce fișiere sunt *staged* utilizând comanda:

```
C:\firstrepo> git status
```

Atunci când sunteți siguri că doriți să integrați modificările într-un nou commit, folosiți comanda:

```
C:\firstrepo> git commit -m "subiect"
```

Unde `subiect` conține o scurtă descriere (de preferat de cel mult 50 de caractere) a commitului. Se obișnuiește ca subiectul să fie redactat în modul imperativ, spre exemplu "Fix bug at line 202", "Optimize memory usage in function myFunction", "Add function to compute shortest path", sau "Modify function myFunction to include NULL case".
După ce un commit a fost creat, acesta nu se mai modifică și va rămâne în istoricul proiectului. Se poate reveni, însă, la un commit anterior dacă se dorește.

3 Conectarea la un repository remote

GitLab și GitHub sunt două exemple de platforme ce oferă servicii de găzduire pentru software și care folosesc sistemul Git.

3.1 Conectare prin ssh

SSH (Secure Shell) este un protocol de rețea care permite conectarea în siguranță la alte calculatoare. Autentificarea se realizează printr-un sistem de criptare asimetric bazat pe o pereche de chei: una publică (folosită pentru a cripta mesajele) și alta privată (folosită pentru a decripta mesajele).
Pentru a vă crea propria pereche de chei ssh, utilizați comanda:

```
C:\> ssh-keygen
```

Vi se va cere să alegeți un director în care să salvați cele două chei. Pentru a evita setări suplimentare ulterioare, alegeți directorul default, apăsând **Enter**. Implicit, directorul este `C:\Users\nume_utilizator\.ssh`. În funcție de setările sistemului de operare, acest director ar putea fi *hidden*.

Aceasta generează două fișiere în directorul ales: unul conținând cheia publică, ce are extensia `.pub` și altul, cu aceași denumire aleasă, dar fără extensie, conținând cheia privată.

Pentru a crea conexiunea cu serverul GitLab sau GitHub trebuie să adăugați cheia publică în contul vostru (să copiați conținutul fișierului cu cheia publică):

În GitHub, la Profile → Settings → SSH and GPG keys → New SSH key.
În GitLab, la Profile → SSH keys → Add a SSH key.
Dacă lucrați pe un calculator din laborator și lăsați cheia privată SSH după terminarea laboratorului, oricine se va putea face modificări în contul vostru remote. Asigurați-vă că la terminarea orei ștergeți perechea de chei din calculator.

3.2 Repository remote

Până acum ați creat un repository local. Este nevoie să creați și unul pe server, apoi să le sincronizați pe cele două.

Pentru a crea un repository nou,
în GitHub, la Profile → Your repositories → New.
în GitLab, la New Project → Create Blank Project.
În ambele variante, asigurați-vă că ați debifat opțiunea **Initialize repository with a README**. Aceasta trebuie debifată deoarece urmează să încărcați pe server un repository local deja existent. Fiecare proiect trebuie să conțină, însă, un fișier README, așadar îl veți crea voi.

Pentru a vă putea conecta repository-ul local la cel de pe server, specificați prin comanda `git remote` adresa repository-ului remote:

```
C:\firstrepo> git remote add origin https://github.com/user/repo.git
```

Atât în GitHub, cât și în GitLab, adresa o obțineți navigând în repository, apoi Code → (Clone with) SSH.

După ce ați executat comanda de mai sus, `origin` va fi un *alias* pentru adresa `https://github.com/user/repo.git` și îl veți putea folosi de fiecare dată când vreți să vă referiți la această adresă pentru a trimite modificările pe server sau a le prelua de acolo. În acest moment repository-ul vostru local este conectat la cel de pe server. Puteți avea mai multe copii locale (pe mai multe calculatoare) ale repo-ului remote, iar ele pot fi sincronizate cu ajutorul următoarelor comenzi.

Pentru a obține cea mai nouă versiune a repo-ului de pe server, folosiți comanda `git pull`:

```
C:\firstrepo> git pull origin main
```

Observați specificarea adresei (prin aliasul `origin`) de la care se face preluarea repository-ul. Iar `main` (sau alternativ `master`, în funcție de cum a fost configurat repository-ul) se referă la *branch*-ul de pe care se face copierea.

Pentru a publica pe server commit-utile create local se folosește comanda `git push`:

```
C:\firstrepo> git push origin main
```

Ordinea acestor comenzi este importantă atunci când lucrați cu un repository remote, în special atunci când colaborați cu alți programatori. Înainte de a trimite pe server modificări (cu `push`) trebuie să vă asigurați că aveți ultima versiune a programului, altfel commit-urile pe care încercați să le trimiteți vor fi în urma celor deja existente pe server, situație care se numește conflict. Într-o anumită măsură, conflictele sunt inevitabile dacă lucrați în echipă și mai mulți programatori modifică în același timp același program sau aceeași secvență de cod. Există un mecanism automat de rezolvare a conflictelor, însă în cazul în care acesta eșuează, ele trebuie rezolvate manual. Pentru a evita situația în care există commit-uri noi în repository-ul remote, pe care nu le aveți local în momentul în care vreți să trimiteți modificări pe server, faceți `pull` înainte de `push`.

Probleme

În laborator

1. Dacă lucrați de pe calculatorul personal și nu aveți instalat Git, urmați instrucțiunile din Secțiunea 1.

2. Creați-vă un cont pe Github sau Gitlab.

3. Creați un director pentru laboratorul de astăzi (nu pe Desktop), în care copiați fișierul `lab2.c`.

4. (0.25p) Inițializați git în acest director, urmând instrucțiunile din Secțiunea 2.1.

5. (0.5p) Creați un commit prin care includeți fișierul `lab2.c` în repo-ul pe care l-ați creat anterior. Urmăriți instrucțiunile din Secțiunea 2.2.

6. (0.25p) Creați o pereche de chei SSH și copiați cheia publică în contul vostru Github/Gitlab, după indicațiile din Secțiunea 3.1.

7. (0.5p) Creați un repository nou pe serverul Github/Gitlab și adăugați adresa în alias-ul origin în repo-ul local, urmărind primele instrucțiuni din Secțiunea 3.2. NU faceți `pull` sau `push` încă.

8. (0.5p) În acest moment repo-ul local conține un commit, în timp ce repo-ul remote nu conține niciun commit. Cu alte cuvinte, cel local este înaintea celui remote, deci puteți face push fără a risca un conflict. (Acesta este motivul pentru care a fost necesar să nu inițializați repo-ul remote cu un README - echivalent cu un commit pe server). Aceasta este singura dată când puteți face `push` (pentru a trimite modificări pe server), înainte de a face `pull` (pentru a prelua ultima versiune de pe server).
   Folosiți comanda `push` pentru a trimite pe server conținutul repo-ului local, urmând instrucțiunile din Secțiunea 3.2. Verificați în interfața Github/Gitlab că vedeți commit-ul.

9. (0.5p) Fișierul `lab2.c` conține o eroare. Corectați-o, faceți un commit nou cu modificarea, trimiteți-o pe server și verificați că vedeți și remote modificările.