

Imaginați-vă următoarea situație: atunci când ați construit un proiect, ați scris câteva linii de cod. Ați terminat partea dvs. de lucru și colegul dvs decide să scrie câteva rânduri de cod ale lui. Mai târziu, atât dvs., cât și colegul dvs. faceți câteva modificări pe codul existent. Se constată că există anumite erori, dar până acum a devenit deja prea complicat să se determine cine a făcut modificările problematice și când. Cum să faceți acest sistem cât mai transparent posibil și ușor de urmărit – acesta va fi subiectul prezentat în această lecție.

De asemenea, în timp ce lucrați la proiect, probabil veți salva fișierele pe calculatorul dvs. Orice modificare a unuiu dintre aceste fișiere l-ar suprascrie pe cel precedent și nu ar fi posibil să se revină la starea anterioară. Git permite și revenirea la punctele înregistrate anterioare (checkpoint) și urmărirea modificărilor și permite colaborarea.

Termenul de versiune a codului

Inima fiecărui produs software este codul lui sursă. Astfel, se poate spune că fără codul sursă al limbajelor de programare corespunzătoare, aplicația nici nu poate exista.

Pe măsură ce, în timpul dezvoltării, numărul de linii de cod care alcătuiesc un site sau orice aplicație crește, este de înțeles că devin din ce în ce mai dificil de urmărit modificările care se fac codului. Dacă la aceasta se adaugă și munca în echipă, care presupune actualizarea aceluiași cod de către un număr mai mare de programatori, situația devine și mai complicată. Pentru a se face ordine în munca de programare și pentru a facilita întreținerea, au fost create de-a lungul timpului numeroase sisteme de versionare a codului (*VCS – version control systems*). Pe lângă această denumire de bază, astfel de sisteme se mai numesc și sisteme *revision control* și *source control*.

Versiunea codului este abordarea care permite urmărirea modificărilor codului sursă. Mai simplu spus, versiunea permite ca fiecare modificare a codului să fie înregistrată și să permită returnarea la o astfel de versiune în orice moment. Pentru a înțelege mai bine acest lucru, vom

analiza un exemplu. Clientul unei aplicații solicită modificarea unei anumite funcționalități, care necesită corectarea codului sursă. Codul existent este înlocuit cu cel nou, iar apoi clientul decide că vrea să întoarcă funcționalitatea veche înapoi în aplicație. Dacă nu se folosește un sistem de versiuni, programatorului nu-i rămâne nimic altceva de făcut decât să scrie din nou codul funcționalității vechi. Pe de altă parte, cu un sistem de versiuni, procesul de revenire la o versiune veche implică doar câteva clicuri.

Folosind un sistem de versiuni, programatorii pot accesa în orice moment informații importante despre istoricul proiectului, ce cuprinde:

- toate modificările care au fost făcute;
- cine le-a făcut;
- când au fost făcute;
- de ce au fost făcute.

Astăzi, versiunea se realizează folosind numeroase sisteme, respectiv programe de calculator.

Ce este Git?

Cel mai cunoscut sistem de versiune de astăzi este cu siguranță Git. Este vorba de sistemul creat de creatorul nucleului Linux - Linus Torvalds, în timp ce lucra la cel mai popular produs al său. Datorită simplității sale, vitezei și faptului că este un software open source, Git a câștigat o popularitate mare. Este folosit intens cu ocazia dezvoltării web, pentru proiecte de diferite dimensiuni, de la site-uri simple în care lucrează un programator până la sisteme web complexe, ce implică echipe mari de dezvoltare.

Git permite controlul modificărilor în fiecare din versiuni, backup-ul proiectului și partajare. Scopul Git-ului este de a îmbunătăți gestionarea lucrului la proiect și a modificărilor în timpul lucrului.

Toate informațiile ce sunt salvate folosind Git sunt stocate într-un

depozit. **Depozitul** reprezintă un director, respectiv un folder în care se află toate versiunile trimise către Git și, de fapt, noi trimitem anumite checkpoints către Git în timpul creării proiectului; acestea se numesc **commits**.

Acest lucru înseamnă că plasăm, de fapt, commits în depozitul Git, iar fiecare dintre ele are propriul **identificator**. De exemplu, de fiecare dată când corectăm o anumită eroare în program sau creăm un element nou, și înainte de a continua este de dorit să creăm și să trimitem un commit cu o descriere a ceea ce am făcut, astfel încât, dacă vom greși ceva în următorii pași, putem să ne întoarcem la acel checkpoint salvat. Acest checkpoint este util și pentru colegi, deoarece pe baza lui știu ce s-a făcut și când.

Depozitul Git poate fi plasat **local**, pe calculatorul dvs., unde în cadrul lui efectuați backup și versiuni, dar în afară de aceasta, este posibil să stocați depozitul Git și într-o locație la distanță, ceea ce este mult mai practic dacă lucrați în echipă, deoarece mai mulți membri vor putea să acceseze acel depozit și să trimită modificări noi și să lucreze la proiect. De asemenea, depozitul de la distanță este convenabil atunci când lucrați independent, deoarece datele sunt stocate pe un server la distanță, așa că dacă de pe calculatorul dvs. s-ar pierde date datorită unei defecțiuni sau ceva similar, ați avea totuși depozitul cu proiectul pe depozitul partajat.

Există diferite servicii care permit utilizarea Git-ului, unele dintre cele mai cunoscute ce se află pe web sunt:

- GitHub;
- GitLab;
- BitBucket;
- SourceForge.

Ele permit plasarea (găzduirea) depozitului pe un server la distanță și accesul ușor.

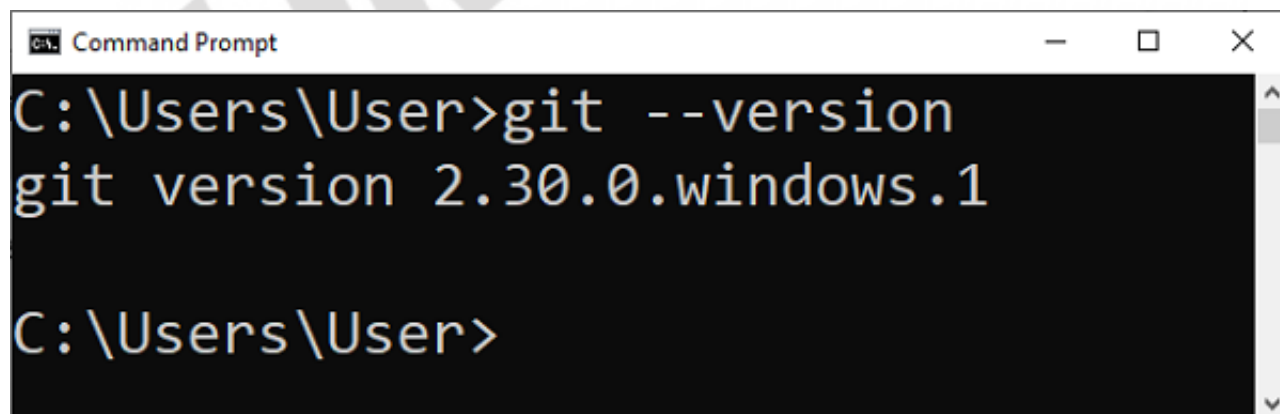
Instalarea și pregătirea pentru lucru

Git este gratuit de utilizat. În funcție de sistemul de operare, procedura lui de instalare diferă și ea.

Înainte de a începe să ne familiarizăm cu Git, este necesar să verificăm dacă pe calculatorul pe care lucrăm este instalat Git. Puteți verifica acest lucru deschizând consola de pe calculator și tastând următoarea comandă:

```
git --version
```

Când confirmați cu un clic pe Enter dacă aveți Git pe calculator, ar trebui să fie listată versiunea lui. În imaginea 3.1. este prezentat un exemplu de rulare a comenzii în consolă și se obțin informații despre versiunea Git ce se află pe calculator.

A screenshot of a Windows Command Prompt window. The title bar reads "Command Prompt". The command prompt shows the path "C:\Users\User>" followed by the command "git --version". The output of the command is "git version 2.30.0.windows.1". The prompt then returns to "C:\Users\User>".

```
C:\Users\User>git --version
git version 2.30.0.windows.1
C:\Users\User>
```

Imaginea 3.1. Verificarea versiunii instalate a Git-ului pe calculator

Windows nu are instrumentul Git instalat în mod implicit și, dacă nu ați primit versiunea Git în consolă, ar trebui să o descărcați și să o instalați:

<https://git-scm.com/download/win>

Pe platformele Linux și macOS, Git este de obicei deja preinstalat, așa că trebuie doar să i se confirme existența cu comanda:

```
git --version
```

Pentru versiunile mai vechi de macOS, Git poate fi descărcat de pe următoarea adresă:

<https://git-scm.com/download/mac>

În cazul în care Git nu există pe Linux, este suficient a se porni următoarea comandă:

```
sudo apt install git-all
```

Notă

Dacă încercați să instalați Git pe sistemele de operare Windows sau macOS utilizând fișierele de instalare aflate tocmai pe adresele web menționate, astfel de fișiere de instalare ar trebui să fie executate după descărcare și trebuie urmărit cursul instalării. Este necesar să se folosească setările implicite de instalare, adică nu este necesar să se facă nicio modificare la setările implicite în timpul instalării. Abia după ce instalarea s-a încheiat, veți putea încerca comenzile Git menționate în continuarea lecției.

Puteți verifica dacă Git a fost instalat cu succes folosind comanda prezentată anterior: `git --version`

Comenzi Git de bază

Git a fost conceput și creat ca un instrument fără interfață cu utilizatorul. De aceea, utilizarea lui presupune utilizarea unei console sau a unui terminal, în funcție de sistemul de operare pe care îl utilizați. Desigur, datorită absenței unui mediu grafic de utilizator, de-a lungul timpului, a fost creat un număr mare de așa-numiți clienți Git, adică programe care permit o utilizare mai simplă a Git-ului. Am decis să încercăm Git în forma sa nativă, astfel încât să puteți înțelege corect modul lui de funcționare.

Modul de funcționare a Git-ului se bazează pe emiterea diferitelor comenzi text. De exemplu, în această lecție am indicat deja o comandă Git, pe care am folosit-o pentru a verifica existența Git-ului, iar aceasta este `git --version`.

Modul în care funcționează Git-ul se bazează pe noțiunea de **depozit**.

Depozitul Git

Depozitul este figura centrală a sistemului Git. Adesea este numit și proiect Git și reprezintă o colecție completă de fișiere și foldere care compun un proiect software - de exemplu, fișierele unui site sau alt tip de aplicație.

Vom crea un depozit Git pe calculator. Pe desktopul dvs. (sau în orice locație), creați un folder gol numit **firstRepo**.

Deocamdată, acesta este un folder simplu care conține un fișier și nu este un depozit Git care ar putea urmări orice fel de modificări.

Crearea unui depozit Git se face prin lansarea următoarei comenzi: git init

Pentru a face folderul nou creat să fie un depozit Git, trebuie să vă **poziționați** în folderul creat folosind comanda cd (dacă a fost creat pe desktop):

```
cd desktop  
cd firstRepo
```

și apoi executați **comanda Git** pentru **crearea depozitului**:

```
git init
```

Dacă totul este în regulă, în consolă va fi imprimat un mesaj cum că a fost creat un depozit Git gol pe calea corespunzătoare (calea diferă în funcție de calculator și de numele folderului și al utilizatorului):

```
Initialized empty Git repository C:/Users/User/Desktop/firstRepo/.git/
```

Această comandă trebuie să fie emisă din folderul în care se dorește a fi creat depozitul.

După crearea depozitului Git, trebuie să îi atribuiți și fișiere. Deschideți acest folder nou creat în mediul Visual Studio Code (Open Folder) și creați un fișier Python nou numit program.py.

Pentru acest pas, desigur, nu este necesară utilizarea mediului Visual Studio Code, este posibil să se adauge și manual fișierul pe calculator (clic dreapta, New text document și numiți-l ca program.py), dar este mai practic cu ajutorul mediului.

Deocamdată, a fost adăugat un fișier nou, dar nu a fost încă înregistrat

în depozit. Pașii pentru înregistrarea fișierelor și modificările vor fi explicate în continuare.

Plasarea fișierelor și folderelor în depozit se face prin implementarea a doi pași numiți **staging** și **committing**. **Staging** poate fi experimentat ca un **pas pregătitor**, iar **committing** ca o acțiune care inițiază **înregistrarea modificărilor** în **depozit**.

Astfel, în depozit sunt înregistrate doar modificările aduse acelor fișiere care se află în starea *staging*, adică pe care a fost efectuată anterior *staging*. Scopul acestui mechanism, care implică doi pași, poate nu-l puteți înțelege în acest moment. Însă fundalul este mai mult decât simplu. Și anume, proiectele constau de obicei dintr-un număr mare de fișiere, ceea ce este cazul și atunci când se dezvoltă site-urile web. Uneori nu veți dori să efectuați modificări în toate fișierele în depozit. În astfel de situații, acest pas intermediar (*staging*) este mai mult decât util, deoarece permite determinarea cu precizie a modificărilor care vor fi introduse în depozit.

Staging, adică pregătirea pentru trimitere, se face folosind comanda `git add`; există două variante:

1. `git add .`
2. `git add yourFileName`

Puteți folosi comanda numărul 1 dacă doriți să pregătiți pentru trimitere **toate** fișierele și **toate** modificările din interiorul depozitului. Puteți folosi comanda numărul 2 dacă doriți să trimiteți doar anumite fișiere și, în acest caz, lângă `git add` specificați numele fișierului dorit. Dacă pregătiți să trimiteți mai multe fișiere în felul acesta, trebuie să repetați această comandă pentru fiecare fișier.

De data aceasta executăm a doua variantă și adăugăm doar un fișier pe care îl avem în folder (în cazul nostru, efectul identic ar fi și cu `git add`, deoarece avem doar un fișier):


```
git add program.py
```

În felul acesta, fișierul program.py este plasat în starea staging, unde totul este pregătit pentru înscrierea lui în depozit.

Commit, respectiv predarea sau finalizarea, se face folosind comanda următoare:

```
git commit -m "Initial Commit"
```

Pe lângă textul comenzii, comanda prezentată conține și o descriere a modificărilor ce au fost făcute codului. Acest lucru facilitează mai târziu găsirea în versiuni diferite ale aceluiași cod.

Crearea unei identități de utilizator

Înainte de înscrierea modificărilor în depozit, este necesar ca Git să aibă date despre utilizatorul care efectuează scrierea. Dacă Git nu are astfel de date, crearea unei identități de utilizator se poate face prin lansarea următoarelor două comenzi:

```
git config --global user.email "you@example.com"  
git config --global user.name "Your Name"
```

Este foarte important să știți că nu se poate efectua niciun commit dacă identitatea utilizatorului nu a fost setată anterior. Deci, dacă vi se cere să definiți o identitate de utilizator după o încercare commit, trebuie să știți că un astfel de commit a eșuat. De aceea, într-o astfel de situație, după definirea identității utilizatorului în modul arătat în rândurile anterioare, este necesar să se repete commit-ul inițial.

Parcurgând pașii de până acum, s-a creat un depozit Git cu un fișier.

Pentru a vedea puterea reală a sistemului Git, este suficient ca, după pașii descriși, să faceți câteva modificări fișierului nostru Python.

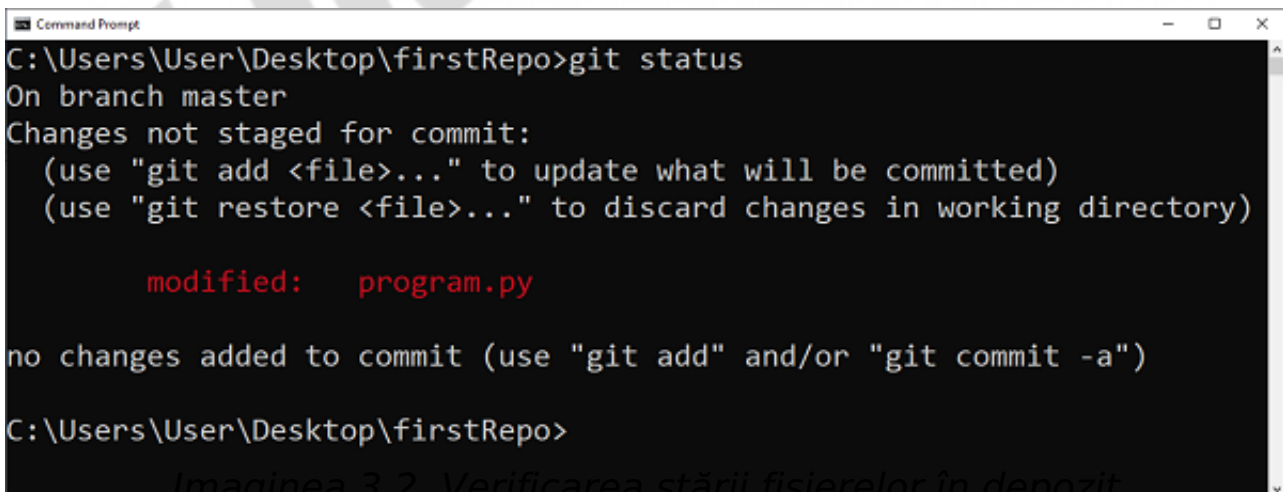
În cadrul fișierului program.py, adăugați următoarea linie de co:

```
print("Hello from my first repository!")
```

După efectuarea modificărilor, salvați fișierul (File -> Save sau Ctrl + S, respectiv Cmd + S).

Git are o comandă pentru verificarea stării fișierelor care sunt scrise în depozit:

```
git status
```



```
Command Prompt
C:\Users\User\Desktop\firstRepo>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)

        modified:   program.py

no changes added to commit (use "git add" and/or "git commit -a")
C:\Users\User\Desktop\firstRepo>
```

Imaginea 3.2. Verificarea stării fișierelor în depozit

În imaginea 3.2. puteți vedea că Git-ul a detectat că există modificări în fișierul program.py, adică că fișierul a fost modificat de la ultimul commit realizat.

Care sunt schimbările? În timpul primului commit, fișierul a fost gol, iar după aceea am adăugat comanda print.


Dacă am dori să scriem acum o astfel de modificare în depozit, ar fi necesar să facem din nou *staging* și *committing*.

```
git add program.py  
git commit -m "Print command added"
```

În sfârșit, Git oferă și o analiză a tuturor acțiunilor commit:

```
git log
```

Prin trimiterea acestei comenzi se obține o vizualizare a tuturor commit-urilor din depozitul curent, cu informațiile însoțitoare (imaginea 3.3.). Commit-urile sunt listate în ordine de la cel mai nou la cel mai vechi.



```
Command Prompt  
C:\Users\User\Desktop\firstRepo>git log  
commit b12b47267df2706fb1b717d14b8e23e681f5b8aa (HEAD -> master)  
Author: User  
Date: Fri Aug 5 12:39:13 2022 +0200  
  
    Print command added  
  
commit 005ee8e1f86013aba49acc096d89cc592509f0bc  
Author: User  
Date: Fri Aug 5 12:30:13 2022 +0200  
  
    Initial commit
```

Imaginea 3.3. Afișajul tuturor commit-urilor

Conceptul de ramificare Git

Pe lângă depozite, ramificarea este unul dintre conceptele de bază ale Git-ului. Și anume, un proiect Git este format dintr-una sau mai multe ramuri (*branches*). Git creează automat o ramură inițială numită ramură master (*master branch*). Programatorul are posibilitatea de a crea un număr arbitrar de ramuri suplimentare. Ideea principală a ramificării este posibilitatea dezvoltării paralele a mai multor versiuni diferite ale unui singur produs software. De exemplu, o ramură poate servi ca principală și alta pentru testare și probare. Git permite crearea de ramuri noi, dar și îmbinarea a două sau mai multor ramuri într-una singură.

Setările fișierelor din cadrul depozitului

Fișierele din interiorul directorului de lucru se pot afla într-una din două stări: urmărite (tracked) și neurmărite (untracked). Fișierele urmărite sunt fișiere ce sunt cunoscute de Git și pot fi nemodificate (unmodified), modificate (modified) sau pregătite (staged). Fișierele neurmărite sunt toate celelalte fișiere. Atunci când este creat pentru prima dată un depozit gol și după aceea sunt adăugate unele fișiere, acele fișiere sunt urmărite până când sunt adăugate în stage și nu comitting, astfel încât depozitul să le recunoască. Dacă este descărcat un depozit existent deja, atunci toate fișierele sunt urmărite, dar nemodificate.

Atunci când se modifică un fișier, acesta trece prin mai multe stări, respectiv etape:

- **Modificat (Modified):**

În această fază sunt aplicate modificările dorite asupra fișierelor (care sunt stocate separat de codul original). Modificările efectuate în această fază nu vor fi aplicate codului original până când nu vor fi trecute prin următoarele două faze.

- **Pregătit (Staged):**

Staging-ul este faza înainte de comitting-ul fișierelor. Aici se

poate face verificarea dacă toate modificările sunt bune înainte de a face fuziunea cu restul codului. În această fază, este posibil să se selecteze fișierele care vor fi pregătite (staged).

- **Committed:**

Aceasta este ultima etapă, în care se aplică toate modificările care au fost făcute. Fiecare commit reprezintă modificările care au fost făcute, împreună cu informațiile despre cine le-a făcut și când. Dacă commit-ul este făcut, înseamnă că modificările codului au fost aplicate.

Funcționalitatea Git

Cele mai importante funcționalități ale Git-ului sunt:

- **Versiunea (versioning):** Funcționalitatea principală a Git-ului (ca și alte VCS) este versiunea. De fiecare dată când unul dintre membrii echipei dvs. face o modificare a unui fișier, Git o înregistrează ca o versiune nouă a fișierului. Dacă în orice moment vedeți o eroare, puteți reveni cu ușurință la ultima versiune despre care știți că funcționa.
- **Ramificarea (Branching):** Git permite, de asemenea, branching-ul, de exemplu, ramificarea. Ceea ce reprezintă, de fapt, ramificarea este posibilitatea ca, pornind de la orice versiune a proiectului dvs., dvs. și colegul dvs. să începeți să priviți problema în moduri diferite. Este posibil să creați două ramuri diferite ale aceluiași proiect. Modificările pe care le faceți ramurii (branch) dvs. nu vor avea niciun efect asupra ramurii de cod a colegului dvs.
- **Fuzionarea (merging):** Când ați terminat de făcut modificări la versiunea dvs. de cod, respectiv la ramura dvs., este posibil să faceți merging, adică fuzionarea diferitelor versiuni. Git vă permite, dacă versiunile sunt compatibile, să uniți versiuni diferite de cod și să adunați ideile la care ați lucrat dumneavoastră și alți membri ai echipei dumneavoastră. Ar trebui să fiți atenți la fuzionare, deoarece pot apărea conflicte dacă programatorii lucrează pe ramuri diferite ale aceluiași

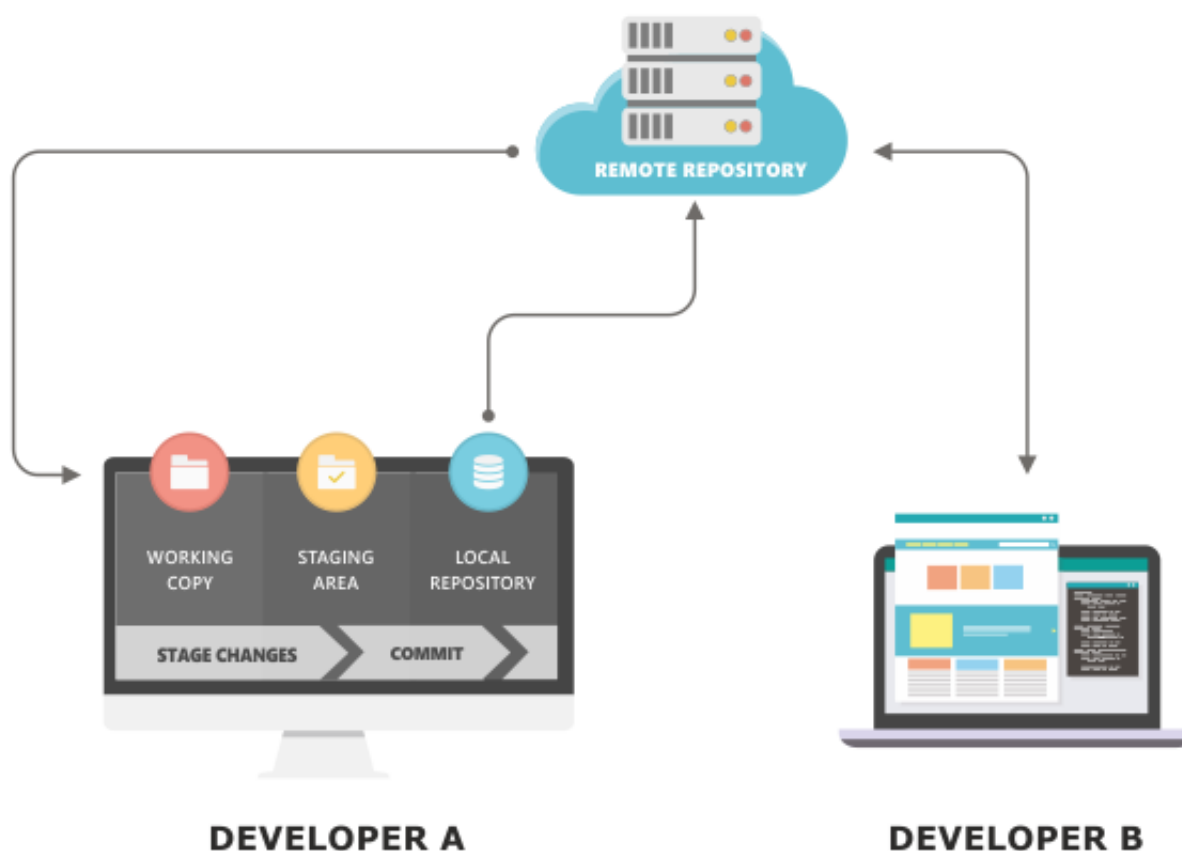
fișier. Membrul echipei care inițiază fuziunea într-o astfel de situație ar trebui să decidă dacă va accepta modificările de la propria ramură sau de la ramura de intrare, pentru a rezolva apariția conflictelor.

- **Păstrarea (safekeeping):** Mulți utilizatori folosesc Git pur și simplu pentru că toate modificările făcute sunt salvate. Acest lucru permite ca tot codul să fie într-un singur loc, sigur.

Ce este GitHub?

Pentru începători pot apărea multe confuzii atunci când termenul GitHub este amestecat în povestea lui Git. Este vorba de fapt de unul dintre cele mai populare sisteme online pentru găzduirea proiectelor Git. Având în vedere că Git, așa cum am menționat mai devreme, este un instrument de consolă fără o interfață grafică cu utilizatorul, GitHub poate fi experimentat ca un wrapper grafic pentru Git, disponibil pe web. Astfel, GitHub permite crearea de depozite la distanță, care sunt accesibile prin web, ceea ce reprezintă o oportunitate excelentă pentru lucrul în echipă.

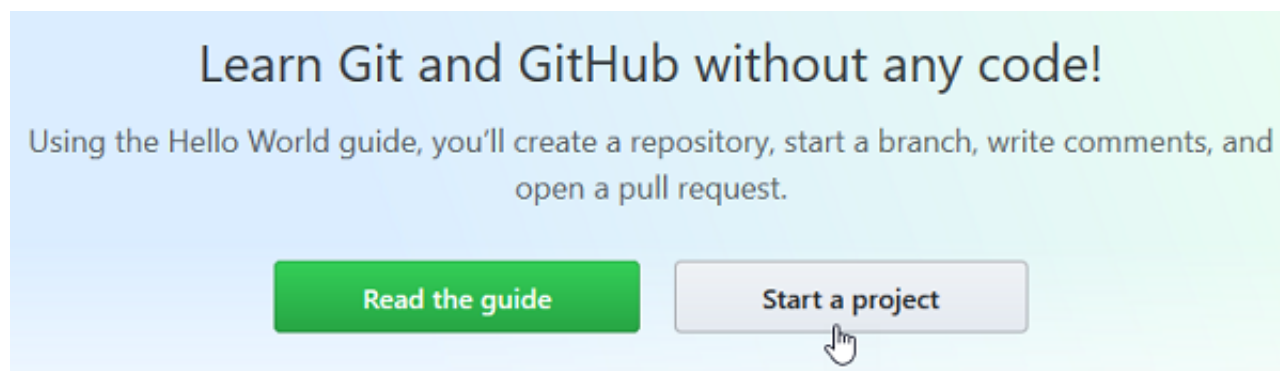
În rândurile anterioare ale acestei lecții a fost ilustrată crearea unui depozit Git local. Totuși, în lucrul în echipă, există în general un depozit central la distanță, în timp ce fiecare dintre programatori are un depozit local pe calculatorul său, așa cum ilustrează imaginea 3.4.



Imaginea 3.4. Principiul de funcționare a depozitelor locale și de la distanță

În felul acesta, programatorii lucrează la depozitele locale și, după ce au făcut modificări codului, trec modificările lor într-un depozit central, care este partajat cu toți ceilalți colegi din echipă. Astfel, toți membrii echipei își încep munca prin sincronizarea depozitului local cu cel central, pentru a se asigura lucrul la cea mai actualizată versiune a codului, care conține toate modificările făcute de restul colegilor din echipă.

Folosirea GitHub-ului presupune crearea în prealabil a unui cont de utilizator. După crearea unui cont de utilizator, trebuie creat un depozit (imaginea 3.5.). Notă: aspectul site-ului web poate diferi ușor, dar funcționalitățile sunt aceleași.



Imaginea 3.5. Opțiunea pentru crearea depozitului

Crearea unui depozit nou pe GitHub necesită o configurare minimă (imaginea 3.6.).

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner *



aleksandralazarevic ▾

Repository name *



repo-python-java



Great repository names are short and memorable. Need inspiration? How about [super-doodle?](#)

Description (optional)



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Imaginea 3.6. Crearea depozitului nou

După crearea depozitului remote pe GitHub, este posibil să se

conecteze depozitul local la cel de la distanță, ceea ce va fi prezentat în continuare.

Înainte de a ne conecta la depozitul de la distanță, GitHub va cere la trimitere introducerea unui nume de utilizator și o parolă. Introduceți acel nume de utilizator pe care l-ați setat în timpul înregistrării, în timp ce pentru parolă introduceți tokenul generat. Este folosit un token în locul unei parole pentru o securitate mai mare, deoarece de îndată ce doriți să lucrați pe un calculator nou, trebuie generat un token nou.

Generarea tokenului

Trebuie să dați clic pe imaginea de profil din colțul din dreapta sus pe GitHub și să selectați **Settings**. În cadrul setărilor din partea stângă, selectați din listă **Developer settings**, apoi **Personal access tokens**.

Cu un clic pe butonul Generate new token, creați un token nou. În câmpul Note este definită o descriere a tokenului. Descrierea este arbitrară și poate fi orice valoare - de exemplu, muncă (work). După aceea, este definită data până când durează tokenul și putem seta ca acesta să nu expire. Din motive de securitate, nu este recomandat să setați valoarea ca tokenul să nu expire, dar dacă lucrați pe un calculator personal și momentan învățați, respectiv exersați, aceasta nu va fi o problemă.

De asemenea, ce este cel mai important, este să setați ce poate face acea persoană cu tokenul și să selectați repo, adică să aibă control deplin asupra depozitelor.

Note

work

What's this token for?

Expiration *

No expiration ⇅ The token will never expire!

GitHub strongly recommends that you set an expiration date for your token to help keep your information secure. [Learn more](#)

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes.](#)

<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events

Imaginea 3.7. Generarea tokenului

După ce confirmați generarea tokenului cu un clic pe butonul Generate token, ca răspuns obțineți o valoare pe care trebuie să o copiați și să o inserați în consolă atunci când vi se cere să introduceți password. Notă: caracterele pentru password nu sunt vizibile în consolă și de aceea poate părea ca și cum nu introduceți nimic, dar totul este în regulă.

Dacă reîmprospătați pagina în browser și doriți să folosiți același token, nu va fi posibil, va trebui să generați unul nou; tocmai în aceasta se

reflectă securitatea prin utilizarea tokenului.

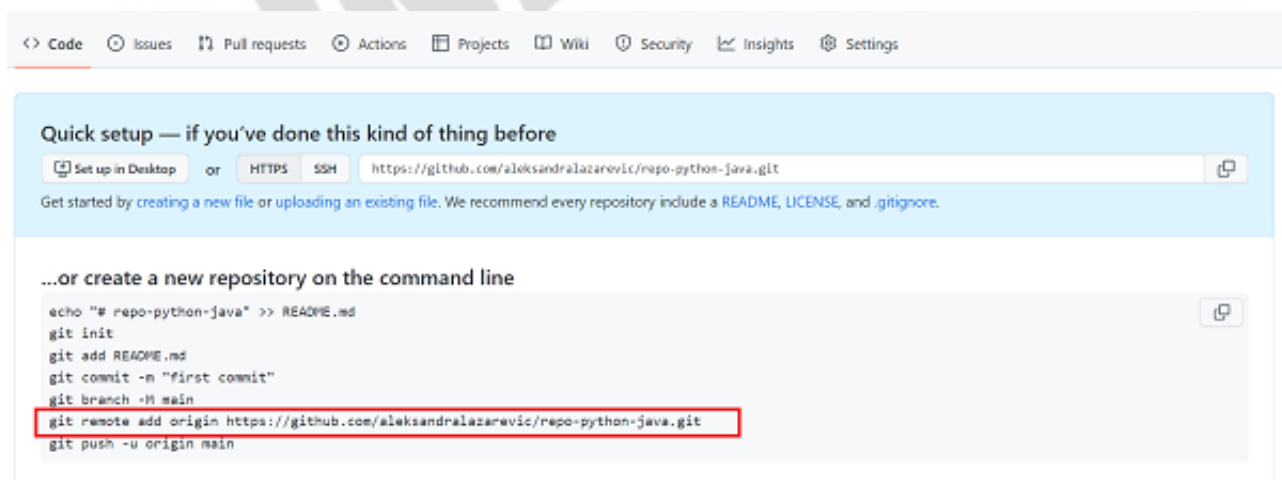
Conectarea unui depozit local cu un depozit GitHub

După crearea depozitului de la distanță și generarea tokenului, se poate face conectarea depozitului local cu cel la distanță nou creat. Acest lucru se face cel mai ușor prin lansarea următoarei comenzi:

```
git remote add origin https://github.com/aleksandralazarevic/repo-python-java.git
```

Comanda este **remote add origin**, în timp ce restul este adresa URL pe care se află depozitul de la distanță pe care l-ați creat, deci adresa dvs. URL va fi diferită.

Comanda prezentată o puteți găsi în cadrul GitHub, după crearea depozitului (imaginea 3.8.).



Imaginea 3.8. Comanda de conectare a depozitului local cu cel de la distanță

Pentru a transfera fișierele din depozitul local în cel la distanță, se folosește comanda push:

`git push -u origin master`

Această comandă inițiază upload, respectiv încărcarea fișierelor locale în depozitul de la distanță.

Mai devreme s-a menționat că începerea lucrului la un proiect la care participă echipa presupune sincronizarea prealabilă a depozitului local cu cel de la distanță. Acest lucru se realizează folosind comanda pull:

`git pull origin master`

Notă

Pentru a putea executa comenzile menționate anterior, este important să fiți poziționat în consolă în interiorul folderului care reprezintă depozitul.

Clonarea unui depozit de la distanță

O altă modalitate de conectare la depozitul dvs. de la distanță sau de descărcare a unui depozit existent este folosirea comenzii `git clone`. Cu ajutorul acestei comenzi se descarcă depozitul de la distanță pe calculatorul dvs.

Când depozitul este descărcat, îl veți vedea pe calculatorul dvs. și sub forma unui folder și puteți face pe el diferite modificări ce sunt necesare. După ce ați terminat cu modificările, le-ați verificat și doriți să le transferați într-un depozit de la distanță, pașii sunt aproape aceiași:

```
git add .  
git commit -m "New changes are..."  
git push
```


Deoarece depozitul este deja conectat la cel remote, dacă trimitem la ramura principală, este suficient să scriem `git push` și fișierele vor fi trimise în depozitul de la distanță.

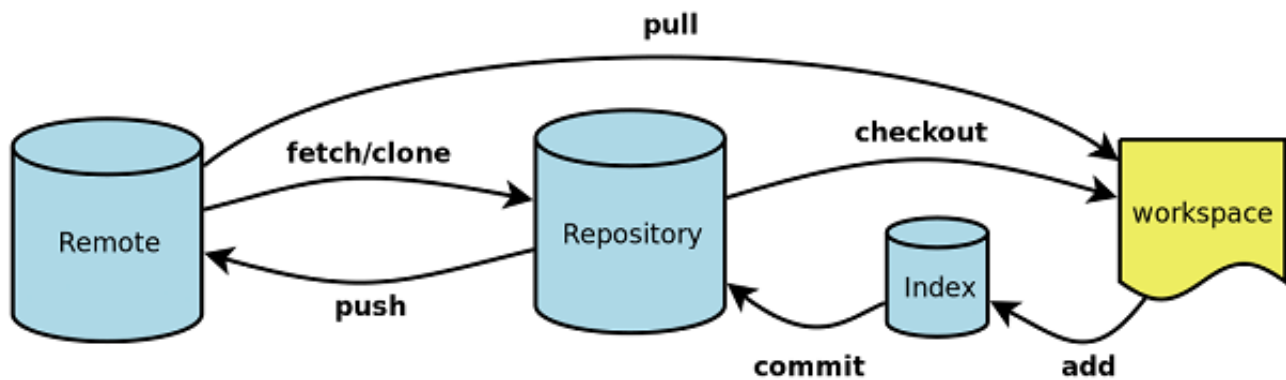
În cadrul conținutului multimedia al lecției, toți pașii menționați anterior sunt descriși în detaliu prin intermediul exemplelor.

Comenzi Git

Acestea sunt câteva dintre cele mai frecvente comenzi pe care le veți întâlni folosind Git în munca de zi cu zi:

Comanda	Descriere
<code>git commit</code>	aplică modificările ce au fost făcute pe fișiere
<code>git push</code>	aplică fișierele commit pe depozitul de la distanță
<code>git pull</code>	descarcă fișiere din depozitul de la distanță
<code>git branch</code>	arată o listă cu toate ramurile
<code>git clone</code>	copiază depozitul de la distanță
<code>git add</code>	adaugă modificări la stage-ul curent
<code>git checkout</code>	permite trecerea la o altă ramură
<code>git merge</code>	unește două ramuri
<code>git remote</code>	verifică ce sursă la distanță este selectată și permite selectarea unei surse noi la distanță

Tabelul 3.1. Cele mai des folosite comenzi Git



Imaginea 3.9. Fluxul utilizării comenzilor git¹

Să ne imaginăm situația în care un coleg partajează cu dvs. un depozit ce se află pe serverele GitHub. Pentru a descărca fișierele din depozit pe calculatorul dvs., trebuie să folosiți comanda `git clone`. În acel moment, pe calculatorul dvs. există un depozit. Dacă doriți să lucrați pe propria ramură, pentru ca codul dvs. să nu se amestece cu codul altor membri ai echipei, trebuie să rulați comanda `git checkout`. Deoarece colegul dvs. a subliniat să creați un fișier nou, pentru ca git să urmărească modificările aduse acestui fișier, trebuie să folosiți comanda `git add`. După ce ați terminat de făcut modificări la fișier, trebuie să salvați modificările pe calculatorul local folosind comanda `git commit`. Dacă doriți să partajați codul înapoi cu echipa dvs., adică să trimiteți codul modificat înapoi la serverele GitHub, trebuie să folosiți comanda `git push`.

Instrumente Git

Pe lângă comenzile de bază prezentate în tabelul 3.1., există și comenzi suplimentare, care nu sunt folosite în munca de fiecare zi, dar permit funcționalități avansate:

- **Stashing and cleaning:** Dacă în timpul lucrului la proiect și modificările codului este necesară trecerea la o altă ramură, iar fișierul în care se fac modificări deocamdată nu este finalizat și gata pentru commit, este posibil să salvați temporar

modificările folosind comanda *git stash*. Toate modificările din stash pot fi accesate în orice moment. Dacă codul aflat în stash nu mai este necesar, acesta poate fi eliminat cu comanda *git clean*. Git vă permite să creați o ramură nouă din stash-ul existent.

- **GPG (GNU Privacy Guard):** GPG permite ca fiecare commit ce este efectuat să provină dintr-o sursă legitimă. La fel ca majoritatea VCS-urilor, Git are posibilitatea de a marca anumite puncte din istoria unui depozit ca fiind importante. Pentru această funcționalitate folosim taguri. Este posibil să configurați Git atunci când la activarea procesului de commit se aplică private key, ce este stocată în taguri. Mai târziu, ceilalți utilizatori pot verifica tagurile și pot vedea dacă acel commit este bun.
- **Searching (pretraga):** Git permite căutarea în depozit în două moduri. Este posibil să folosiți *git grep*, care caută în toate directoarele existente pentru stringul text, sau să căutați logs folosind comanda *git log*, unde putem determina când și unde a fost făcută modificarea.
- **Debugging:** Git are multe instrumente de debugging. Dacă trebuie să se vadă care linie de cod a fost schimbată și de către ce utilizator, se poate folosi comanda *git blame*. Dacă nu se știe în ce secțiune a codului se află eroarea, se poate folosi comanda *git bisect*. Această comandă permite căutarea întregului istoric de commit și facilitează găsirea problemelor.
- **Bundling:** Cu comanda *git bundle*, este posibil să se combine toate modificările într-un singur [fișier binar](#), care apoi poate fi redirecționat cu ușurință către alt utilizator.
- **Reuse recorded resolution:** Comanda *git rerere* permite aplicarea automată a soluției care a fost creată pentru un anumit conflict, de fiecare dată când acel conflict este găsit în cod.
- **Git ignore:** Dacă este necesar ca git să nu documenteze modificările în fișierele individuale, este posibil să se utilizeze **Git ignore**. Este necesar să creați un fișier `.gitignore`, în care să fie definite fișiere sau foldere care nu trebuie documentate. Dacă utilizatorul are nevoie ca fișierul care este documentat în prezent să nu mai fie documentat, trebuie

folosită comanda `git rm --cached`.

Cum diferă Git de alte VCS-uri?

Majoritatea VCS-urilor stochează date ca o listă a tuturor modificărilor făcute fișierelor. Pe de altă parte, de fiecare dată când un utilizator face un commit, Git înregistrează un snapshot al acelui fișier, adică face o fotografie a fișierului în acel moment. Dacă nu s-au făcut modificări în fișier, Git nu salvează fișierul din nou, ci salvează doar linkul către versiunea anterioară a fișierului. Un alt avantaj al Git-ului este că stochează toate fișierele local, ceea ce economisește mult timp și previne întârzierea care ar apărea dacă fișierele ar fi pe server. Toate operațiunile pe care doriți să le efectuați sunt efectuate pe hard diskul local. Atunci când dați push unuia sau mai multor commit-uri, este posibil să stocați fișiere și pe serverele GitHub, unde acestea vor fi disponibile pentru toți membrii echipei. Restul programatorilor pot să dea pull în orice moment, adică pot „descărca” versiunea curentă a fișierului și să o modifice cum doresc.

Interesant

Mascota GitHub-ului este o creatură care este o combinație între o pisică și o caracatiță. A fost creat de designerul Simon Oxley, care a proiectat și pasărea ce se află în logo-ul Twitter. GitHub a numit creația sa Octocat sau Octopisica. Mulți utilizatori ai GitHub-ului care se ocupă de design au contribuit la această creație, desenând sute de variante ale Octocat-ului.



Imaginea 3.10. Variații de design al mascotei GitHub²

-
1. https://illustrated-git.readthedocs.io/en/latest/_images/git-flows.svg
 2. <https://octodex.github.com/>

LINKgroup