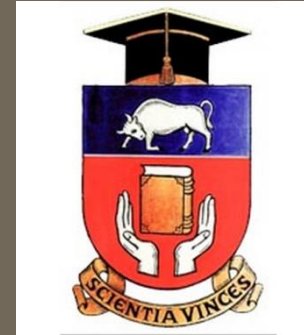


Universitatea de Stat din Tiraspol



GIT și GitHUB

A elaborat:

ANDRIAN DASCAL, masterand, an. II

A verificat:

OLGA CERBU, dr. conf. univ.

Ce este GIT?

- Git este un sistem revision 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, Ruby on Rails, Samba.

Ce este GITHUB?

- **GitHub este un website și un serviciu foarte des adus în discuție de pasionații de IT, însă există și foarte multe persoane care habar nu au ce este și care nu înțeleg ce face de fapt acest serviciu. În continuare vom aduce puțină lumină cu privire la acest aspect.**
- **Cuvântul în sine este compus din două elemente: „Git” și „Hub”. Să vedem la ce se referă fiecare dintre acestea.**
- **Pentru a înțelege mai bine GitHub, trebuie să știi întâi la ce se referă Git. Este un sistem open-source de control a versiunilor conceput de Linus Trovalds – aceeași persoană care a creat sistemul de operare Linux. Git este în fapt asemănător cu alte astfel de sisteme de control a versiunilor, precum Mercurial, Subversion sau CVS.**



Istoria GIT

- Dezvoltarea Git a început după ce mai mulți developeri ai nucleului Linux au ales să renunțe la sistemul de revision control proprietar BitKeeper. Posibilitatea de a utiliza BitKeeper gratuit a fost retrasă după ce titularul drepturilor de autor a afirmat că Andrew Tridgell a încălcat licența BitKeeper prin acțiunile sale de inginerie inversă. La conferința Linux.Conf.Au 2005, Tridgell a demonstrat în timpul discursului său că procesul de inginerie inversă pe care l-a folosit a fost pur și simplu o sesiune telnet pe portul corespunzător al serverului BitKeeper și rularea comenzii *help* pe server.
- Controversa a dus la o renunțarea rapidă la sistemul BitKeeper care a fost înlocuit cu un nou sistem intitulat Git construit special pentru scopul de revision control în cadrul proiectului Linux kernel. Dezvoltarea noului sistem a fost începută de Linus Torvalds în 3 aprilie 2005 pentru a fi anunțat câteva zile mai târziu (aprilie 6) pe lista de email a proiectului Linux kernel. O zi mai târziu, noul sistem a început să fie folosit pentru dezvoltarea actuală de cod pentru proiectul Git. Primele operații merge a avut loc pe data de 18 aprilie. În data de 16 iunie, versiunea 2.6.12 Linux kernel a fost pusă în Git care continuă și în ziua de azi să fie sistemul revision control folosit de proiectul Linux kernel. Tot în această perioadă, și tot cu scopul de a înlocui BitKeeper, a fost creat sistemul Mercurial.

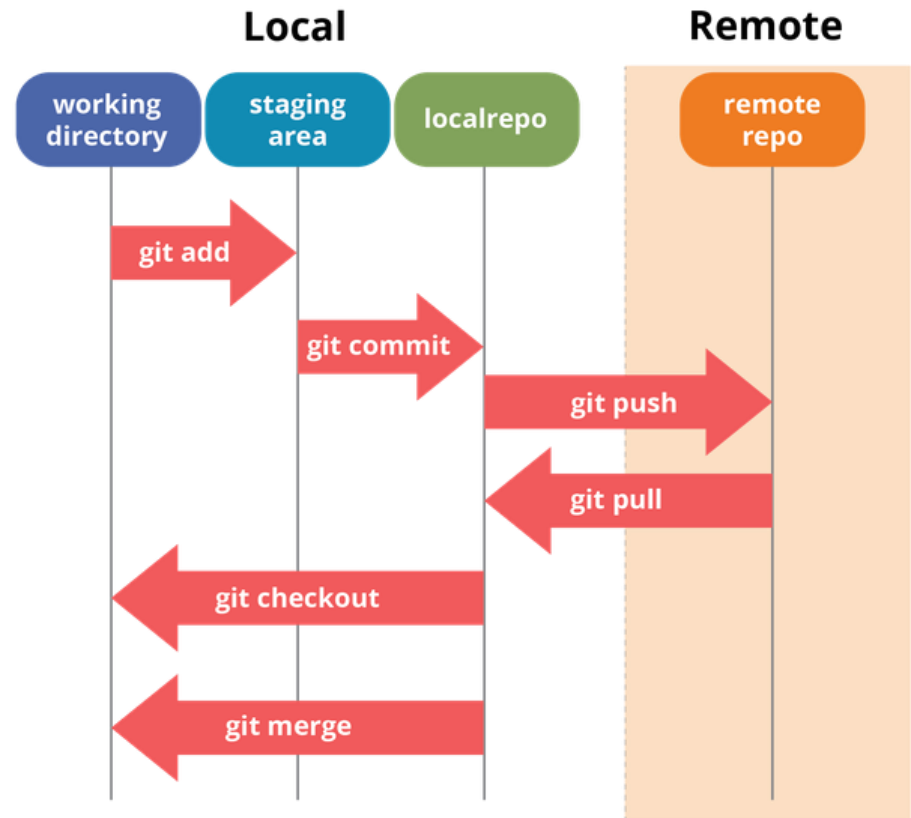
Sistemul de control al versiunilor

- Ce înseamnă asta de fapt? Atunci când web developerii creează ceva (o aplicație, spre exemplu), vor face modificări constante la codul acesteia. Cu fiecare modificare de cod va apărea și o nouă versiune, până când se va ajunge la prima lansare oficială a aplicației respective (după ce a trecut de stadiul de testing/beta).
- Sistemul de control al versiunilor ajută la monitorizarea și centralizarea tuturor modificărilor făcute la un anume proiect. Acest lucru permite web developerilor să colaboreze mult mai eficient, întrucât pot descărca oricând cea mai recentă versiune a software-ului în cauză, îi pot face modificări și pot încărca ulterior versiunea actualizată. Fiecare dezvoltator web poate vedea aceste modificări și își poate aduce la rândul său contribuția.
- În mod similar, chiar și cei care nu sunt implicați deloc în realizarea unui proiect anume pot să descarce fișierele și să le folosească. Majoritatea utilizatorilor Linux sunt familiarizați cu acest proces, întrucât utilizarea Git, Subversion sau a oricărei alte metode similare este o practică destul de comună.

Sistemul de control al versiunilor

De acolo oricine poate descărca fișierele căutate, în special cele necesare pregătirii compilării unui program, a unui software sau a unei aplicații de la nivelul codului sursă.

Dacă vă întrebați de ce Git este alternativa preferată de utilizatori, trebuie să știți că oferă multiple avantaje față de alte astfel de sisteme, inclusiv o modalitate mult mai eficientă de a stoca modificările de fișiere, asigurând în același timp integritatea acestora.



Ce inseamna Hub in GitHub?

- Am lămurit faptul că Git este un sistem de control al versiunilor, similar cu multe altele, dar considerat a fi cea mai bună alternativă. De ce este GitHub atât de special?
- Git este efectiv o unealtă bazată pe linii de comandă, însă locul în care se centralizează toate datele și în care are loc stocarea proiectelor este efectiv Hub-ul, mai exact GitHub.com. Aici dezvoltatorii pot adăuga și stoca proiecte la care lucrează împreună cu alți pasionați.
- De ce este atât de cunoscut GitHub? În continuare vom enumera câteva dintre motive și te vom ajuta să te mai familiarizezi cu câțiva termeni.

Ce inseamna Hub in GitHub?

- *Repository sau Repo* – reprezintă locația unde sunt stocate toate fișierele aferente unui anumit proiect. Fiecare software, proiect sau aplicație va avea un „repo” propriu, ce poate fi accesat cu un URL unic.
- *Fork* – constituie operațiunea ce constă în crearea unui nou proiect pornind de la un altul deja existent. Această facilitățe încurajează dezvoltarea extensivă a oricărui proiect și a multor altor proiecte conexe.
- *Pull request* – în momentul în care ai făcut *fork* pe un *repo* și îți dorești recunoașterea contribuției din partea dezvoltatorilor originali, poți face un *pull request*. Autorii originali ai proiectului îți vor vedea și analiza munca și vor lua decizia dacă o vor accepta sau nu în versiunea finală a proiectului oficial.

GitHUB nu este doar pentru web developeri

- Toată această prezentare te poate duce cu gândul la ideea că GitHub își găsește utilitatea doar pentru programatori. Greșit! Deși e o practică mult mai rară, GitHub poate fi folosit chiar de către tine, în cazul în care ai o echipă care – să spunem – lucrează în comun la editarea unui document Word.
- GitHub poate fi folosit în acest caz ca sistem de control a versiunilor diverse ce necesită actualizare periodică.
- Acum că știi despre ce e vorba, nu îți rămâne decât să începi să testezi performanțele acestui sistem de control a versiunilor.
- Dacă folosești Windows atunci poți intra pe site-ul git și să descarci instaler-ul.

Instalare GIT

- Dar, daca folosesti Linux (Ubuntu) atunci trebuie sa deschizi terminalul (alt+t) si sa scrii urmatoarea comanda:

sudo apt-get install git

- Un repository este de fapt un folder in care git urmareste modificarile. Hai sa facem un folder nou si sa-l numim test

mkdir test

- Acum hai sa intram in acest folder nou

cd test

- Si acum hai sa initializam git

git init

Felicitari! Doar ce ai creat un repository nou!

Configurare GIT

- Nu te grabi sa faci modificari in acest folder. Acum trebuie sa ii spunem lui git numele si adresa de email (pentru configurare).
- Rulati urmatoarea comanda:
git config user.name "Dascal Andrian"
- Rulati urmatoarea comanda pentru email
git config user.email andrusha2204@gmail.com



Modificari in repository

Acum ca ne-am configurat repository-ul putem incepe sa folosim git. Hai sa facem niste mici modificari in acest folder.

Sa facem un fisier nou:

touch fisier1 fisier2

Comanda de mai sus a creat doua fisiere noi numite fisier1 si fisier2

O alta comanda folositoare este aceasta:

git status

Aceasta iti afiseaza statusul repository-ului unde iti arata ce fisiere au fost editate, ce fisiere au fost sterse si ce fisiere au fost create.

In clipa aceasta ne arata cele 2 fisiere create putin mai devreme, fisier1 si fisier2, iar pentru a putea urmari modificarile dina ceste fisiere trebuie sa le adaugam in staging:

git add fisier1

git add fisier2

Acum, daca scriem din nou git status putem observa ca fisierele sunt in staging si putem da commit.

Modificari in repository

Putem da commit folosind comanda git commit:

git commit -m "Am creat doua fisiere noi"

Felicitari! Doar ce ai creat primul commit.

Pentru a vedea commit-urile nu scriem git status ci git log:

git log

Acum hai sa stergem unul dintre aceste fisiere:

rm fisier1

Aceasta comanda a sters fisierul numit fisier1 din folderul test.

Daca scriem din nou git status putem observa ca fisierul1 lipseste (evident, nu?) si va trebui sa updat-am si zona de staging folosind comanda:

git rm fisier1

Acum daca scriem iar git status observam ca fisierul a fost sters si putem face din nou commit:

git commit -m "fisierul1 a fost sters"

9 de comenzi de baza pentru GIT

Iată o listă cu cele mai folosite 9 comenzi elementare pentru Git:

1. **git init** - creează un repo local nou
2. **git clone** - clonează un repository remote în copie locală
exemplu.: `git clone username@host:/path/to/repo`
3. **git add** - adaugă un fișier în indexul repositoryului
exemplu.: `git add <filename.ext>`
4. **git commit** - salvează schimbările în head
exemplu.: `git commit -m "descriptive message"`
5. **git push** - trimite fișierele schimbate în ramura master din repositoryul remote
exemplu.: `git push origin master`
6. **git status** - afișează lista cu fișiere modificate și cu cele care necesită o acțiune de commit sau push
7. **git remote add origin <server>** - dacă nu s-a conectat copia locală la un server atunci se adaugă un server pentru a se putea face push spre acesta
8. **git pull** - descarcă și unește modificările din repositoryul remote cu cele din copia de lucru locală
9. **git diff** - afișează posibilele conflicte și diferențe

20 de comenzi zilnice pentru GIT

- Comenzile de bază pentru lucrul cu repozitoriile se folosesc de cei care dețin un repozitoriu - adică practic de toată lumea pentru că orice director git este un repozitoriu.
- În afară de aceasta, sînt esențiale comenzile pentru lucrul individual al programatorilor pentru oricine face un commit (transmitere), chiar și pentru cineva care lucrează singur.
- Dacă lucrați în echipă, veți avea nevoie de comenzile care sînt listate de asemenea în secțiunea Comenzi pentru lucrul în echipă al programatorilor.
- Integratorii au nevoie de cîteva comenzi în plus la cele de mai sus.
- Comenzi de administrare a repozitoriului sunt pentru administratorii de sistem responsabili pentru păstrarea și transmiterea repozitoriilor git.

20 de comenzi GIT

Comenzi de bază pentru lucrul cu repozitoriile

Aceste comenzi sunt folosite pentru mentenanța repozitoriilor git.

- [git-init\(1\)](#) sau [git-clone\(1\)](#) pentru a crea un repozitoriu nou.
- [git-fsck\(1\)](#) pentru a verifica dac repozitoriul conține erori.
- [git-gc\(1\)](#) pentru a optimizarea și compactarea repozitoriului.

Exemple

Verificarea și curățarea repozitoriului.

```
$ git fsck (1)  
$ git count-objects (2)  
$ git gc (3)
```

1. Executat fără `--full` este de obicei mai ieftin ca consum de resurse și asigură repozitoriului o stare de "sănătate" destul de bună.
2. Enumeră obiecte orfane și spațiul consumat de ele.
3. Sterge obiectele orfane și efectuează și alte operațiuni de optimizare a repozitoriului.

Comenzi pentru lucrul individual al programatorilor

Un programator individual nu face schimb de patch-uri cu alții, și lucrează singur într-un singur repozitoriu, folosind următoarele comenzi.

- [git-show-branch\(1\)](#) pentru a vedea ramificarea curentă a directorului git.
- [git-log\(1\)](#) pentru a vedea evoluția directorului git.
- [git-checkout\(1\)](#) și [git-branch\(1\)](#) pentru a schimba ramificarea directorului git.
- [git-add\(1\)](#) pentru a lucra cu index-ul git.
- [git-diff\(1\)](#) și [git-status\(1\)](#) pentru a vedea starea fișierelor din director.
- [git-commit\(1\)](#) înregistrează modificările curente ale fișierelor în git.
- [git-reset\(1\)](#) anulează schimbările necomise.
- [git-checkout\(1\)](#) (cu parametrii căii de acces) pentru a anula schimbările.
- [git-merge\(1\)](#) pentru a combina modificările a două sau mai multe ramificații.
- [git-rebase\(1\)](#) schimbă ramificarea de bază a directorului git
- [git-tag\(1\)](#) marchează starea curentă a directorului git.

Exemple

Folosirea unei arhive ca punct de start pentru un repository nou.

```
$ tar xzf frotz.tar.gz
$ cd frotz
$ git-init
$ git add . (1)
$ git commit -m "Imaporaarea directorului frotz."
$ git tag v2.43 (2)
```

1. se inițiază un repository git nou în directorul curent
2. se marchează starea curentă a directorului cu eticheta "v2.43"

Lucrul cu ramificările

```
$ git checkout -b alsa-audio (1)
$ edit/compile/test
$ git checkout -- curses/ux_audio_oss.c (2)
$ git add curses/ux_audio_alsa.c (3)
$ edit/compile/test
$ git diff HEAD (4)
$ git commit -a -s (5)
$ edit/compile/test
$ git reset --soft HEAD^ (6)
$ edit/compile/test
$ git diff ORIG_HEAD (7)
$ git commit -a -c ORIG_HEAD (8)
$ git checkout master (9)
$ git merge alsa-audio (10)
$ git log --since='3 days ago' (11)
$ git log v2.43.. curses/ (12)
```

1. Crează o ramificare nouă a directorului.
2. Întoarce `curses/ux_audio_oss.c` la starea inițială
3. Notificați git că ați adăugat un fișier nou; ștergerile și modificările vor fi înregistrate dacă rulați comanda `git commit -a` mai târziu.
4. Arată diferența față de HEAD
5. Trimite.
6. Anulați ultima trimitere, lăsând ce este în ramificarea curentă.
7. Arată diferența de la anularea ultimei trimiteri.
8. Retrimite ceea ce s-a anulat la pasul precedent folosind mesajul original.
9. Treceți la ramura principală.
10. Combină ramificarea creată la primul pas cu ramificare principală.
11. Arată trimiterile (commit); alte moduri de selectare a înregistrărilor pot fi combinate: `--max-count=10` - arată ultimile 10 trimiteri, `--until=2005-12-10`, etc.
12. Arată schimbările de la `v2.43` înapoi care au afectat fișierele din directorul `curses/`.

Comenzi pentru lucrul în echipă al programatorilor

Fiecare programator dintr-o echipă colaborează cu ceilalți membri ai echipei folosind comenzi adăugătoare celor [pentru lucrul individual al programatorilor](#).

- [git-clone\(1\)](#) copiază un repozitoriu într-un director.
- [git-pull\(1\)](#) and [git-fetch\(1\)](#) de la actualizează repozitorul curent cu alt repozitor sau altă ramificare.
- [git-push\(1\)](#) trimite repozitorul curent spre un altul.
- [git-format-patch\(1\)](#) pregătește un patch pentru trimiteră prin e-mail

Exemple

Ia o copie a repozitoriului și trimite modificările înapoi.

```
$ git clone git://git.kernel.org/pub/scm/.../torvalds/linux-2.6 my2.6
$ cd my2.6
$ edit/compile/test; git commit -a -s (1)
$ git format-patch origin (2)
$ git pull (3)
$ git log -p ORIG_HEAD.. arch/i386 include/asm-i386 (4)
$ git pull git://git.kernel.org/pub/.../jgarzik/libata-dev.git ALL (5)
$ git reset --hard ORIG_HEAD (6)
$ git gc --prune (7)
$ git fetch --tags (8)
```

1. Ciclu normal de dezvoltare repetat de una sau mai multe ori.
2. Extrage modificările în formă de patch pentru a fi trimise pe e-mail.
3. `git pull` actualizează ramura curentă.
4. Verifică ce s-a actualizat în directorul indicat.
5. Aduce dintr-o ramură anume a unui anume repozitoriu și combină cu repozitoriul curent.
6. Anulează operațiunea precedentă.
7. Curăță rămășițele de la operațiunea precedentă.
8. Actualizați periodic lista de tag-uri oficiale. Ele se stochează în `.git/refs/tags/`.

Trimiteți în alt repozitoriu.

```
satellite$ git clone mothership:frotz frotz (1)
satellite$ cd frotz
satellite$ git config --get-regexp '^(remote|branch)\.' (2)
remote.origin.url mothership:frotz
remote.origin.fetch refs/heads/*:refs/remotes/origin/*
branch.master.remote origin
branch.master.merge refs/heads/master
satellite$ git config remote.origin.push \
    master:refs/remotes/satellite/master (3)
satellite$ edit/compile/test/commit
satellite$ git push origin (4)

mothership$ cd frotz
mothership$ git checkout master
mothership$ git merge satellite/master (5)
```

1. Copiază repozitoriul `frotz` de pe mașina `mothership`.
2. Clonarea face aceste setări ale variabilelor implicit (by default). Configurează `git pull` pentru a aduce și stoca ramificările de pe `mothership` în `remotes/origin/*`.
3. Configurează `git push` să trimită modificările din ramificarea locală `master` în `remotes/satellite/master` pe `mothership`.
4. Modificările curente vor fi trimise în `remotes/satellite/master` pe `mothership`. Această manevră poate fi folosită ca o metodă de back-up.
5. Pe mașina `mothership`, fuzionați lucrul terminat pe `satelit` în ramificarea principală.

Crearea unei ramificări marcate cu o etichetă

```
$ git checkout -b private2.6.14 v2.6.14 (1)
$ edit/compile/test; git commit -a
$ git checkout master
$ git format-patch -k -m --stdout v2.6.14..private2.6.14 |
  git am -3 -k (2)
```

1. Crează o versiunea privată a repozitoriului.
2. Trimite toate schimbările din ramura `private2.6.14`, în ramura `master` fără o "fuziune" formală.

Comenzi pentru integrator

Integratorul într-un proiect primește schimbările făcute de echipă, le revizuieste, le integrează și publică rezultatul. În afară de comenzile folosite de ceilalți, el mai are nevoie de următoarele comenzi.

- [git-am\(1\)](#) Extrage și integrează patch-urile primite pe e-mail.
- [git-pull\(1\)](#) Aduce modificările de la alți membri și le integrează în ramificarea curentă.
- [git-format-patch\(1\)](#) Pregătește patch-urile pentru a fi trimise pe e-mail.
- [git-revert\(1\)](#) Anulează un commit în caz de necesitate.
- [git-push\(1\)](#) Trimite/publică ramificare curentă.

Exemple

Comenzi de zi cu zi pentru integrator

```
$ git status (1)
$ git show-branch (2)
$ mailx (3)
& s 2 3 4 5 ./+to-apply
& s 7 8 ./+hold-linus
& q
$ git checkout -b topic/one master
$ git am -3 -i -s -u ./+to-apply (4)
$ compile/test
$ git checkout -b hold/linus && git am -3 -i -s -u ./+hold-linus (5)
$ git checkout topic/one && git rebase master (6)
$ git checkout pu && git reset --hard next (7)
$ git merge topic/one topic/two && git merge hold/linus (8)
$ git checkout maint
$ git cherry-pick master~4 (9)
$ compile/test
$ git tag -s -m "GIT 0.99.9x" v0.99.9x (10)
$ git fetch ko && git show-branch master maint 'tags/ko-*' (11)
$ git push ko (12)
$ git push ko v0.99.9x (13)
```

1. Văd starea curentă a repoziitoriului.
2. Văd ramificările curente.
3. Verific cutia poștală și repartizez patch-urile primite.
4. Le aplic interactiv și adaug semnătura mea.
5. Creez o ramificare, aplic cealaltă parte de patch-uri și adaug semnătura mea.
6. Trec una din ramificările locale pe pe ramificarea de bază.
7. Trec pe ramificarea `pu` la starea `next`.
8. Combin ramificările pregătite.
9. Aduc un fix important dintr-o versiune anterioară.
10. Fac o etichetă.
11. Mă asigur că nu am întors accidental ramura `master` dincolo de ce era deja publicat.
Ramura `ko` e o scurtătură pentru ce am în repoziitoriul meu la `kernel.org`, și arată cam așa:

```
$ cat .git/remotes/ko
URL: kernel.org:/pub/scm/git/git.git
Pull: master:refs/tags/ko-master
Pull: next:refs/tags/ko-next
Pull: maint:refs/tags/ko-maint
Push: master
Push: next
Push: +pu
Push: maint
```

În lista afișată de `git show-branch`, `master` ar trebui să aibă tot ce are `ko-master`, și `next` ar trebui să aibă tot ce are `ko-next`.

12. Publică cea mai recentă versiune.
13. Publică și eticheta creată.

Administrarea repoziatoriului

Administratorul repoziatoriului folosește comenzile următoare pentru a seta menține accesul la repozițoriu pentru brogramatori.

- [git-daemon\(1\)](#) pentru a permite accesul la repozițoriu.
- [git-shell\(1\)](#) poate fi folosit ca shell SSH pentru access-ul la repozițoriu.

[update hook howto](#) este un exemplu bun de menținere a unui repozițoriu.

Exemple

Se presupune că există următoarea linie în /etc/services

```
$ grep 9418 /etc/services
git          9418/tcp          # Git Version Control System
```

Lansați serverul git din inetd pentru publicarea directorului /pub/scm.

```
$ grep git /etc/inetd.conf
git      stream tcp      nowait  nobody \
        /usr/bin/git-daemon git-daemon --inetd --export-all /pub/scm
```

(De fapt e o singură linie)

Lansați serverul git din xinetd pentru publicarea directorului /pub/scm.

```
$ cat /etc/xinetd.d/git-daemon
# default: off
# description: Serverul git oferă acces la repozitoriile git
service git
{
    disable = no
    type          = UNLISTED
    port          = 9418
    socket_type   = stream
    wait          = no
    user          = nobody
    server        = /usr/bin/git-daemon
    server_args   = --inetd --export-all --base-path=/pub/scm
    log_on_failure += USERID
}
```

Acesta este configurarea pentru Fedora. Vedeți documentația xinetd(8) pentru alte sisteme de operare.

Dați acces de push/pull pentru programatori.

```
$ grep git /etc/passwd (1)
alice:x:1000:1000::/home/alice:/usr/bin/git-shell
bob:x:1001:1001::/home/bob:/usr/bin/git-shell
cindy:x:1002:1002::/home/cindy:/usr/bin/git-shell
david:x:1003:1003::/home/david:/usr/bin/git-shell
$ grep git /etc/shells (2)
/usr/bin/git-shell
```

1. Shell-ul de log-in e setat la `/usr/bin/git-shell`, care nu permite nimic în afară de `git push` și `git pull`.
2. în multe distribuții `/etc/shells` trebuie să conțină căile spre shell-uri.

Repozitoriu în stil CVS.

```
$ grep git /etc/group (1)
git:x:9418:alice,bob,cindy,david
$ cd /home/devo.git
$ ls -l (2)
lrwxrwxrwx  1 david git    17 Dec  4 22:40 HEAD -> refs/heads/master
drwxrwsr-x  2 david git 4096 Dec  4 22:40 branches
-rw-rw-r--  1 david git   84 Dec  4 22:40 config
-rw-rw-r--  1 david git   58 Dec  4 22:40 description
drwxrwsr-x  2 david git 4096 Dec  4 22:40 hooks
-rw-rw-r--  1 david git 37504 Dec  4 22:40 index
drwxrwsr-x  2 david git 4096 Dec  4 22:40 info
drwxrwsr-x  4 david git 4096 Dec  4 22:40 objects
drwxrwsr-x  4 david git 4096 Nov  7 14:58 refs
drwxrwsr-x  2 david git 4096 Dec  4 22:40 remotes
$ ls -l hooks/update (3)
-r-xr-xr-x  1 david git 3536 Dec  4 22:40 update
$ cat info/allowed-users (4)
refs/heads/master      alice\|cindy
refs/heads/doc-update  bob
refs/tags/v[0-9]*      david
```

1. Pune programatorii în grupul `git`.
2. Și dă acces de scriere în repozitoriu pentru acest grup.
3. Folosiți exemplul `update-hook` al lui Carl din `Documentation/howto/` pentru a stabili politica de acces la ramificații.
4. `alice` și `cindy` pot publica în `master`, numai `bob` poate publica în `doc-update`. `david` este managerul de release și este unica persoană care poate crea și publica tag-uri.

Server HTTP pentru transfer simplu.

```
dev$ git update-server-info (1)  
dev$ ftp user@isp.example.com (2)  
ftp> cp -r .git /home/user/myproject.git
```

1. Asigurați-vă că `info/refs` și obiectele/`info/pachete` sînt actualizate.
2. Încărcați directorul pe serverul HTTP.

Cont GitHub

1. Crearea contului GitHub

Primul pas este crearea unui cont **GitHub**, completând un username, adresa de email și o parolă. La următorul pas va trebui să selectați tipul de cont dorit. Implicit este selectat contul gratuit, care permite doar crearea de repository-uri publice. Apăsați „**Finish sign up**”.

Acesta este un moment bun pentru a valida adresa de email aleasă: tot ce trebuie să faceți este să accesați link-ul din interiorul email-ului primit de la GitHub.

2. Crearea unui repository

Pagina de start a GitHub va conține acum rubrica ***GitHub Bootcamp***, o colecție de resurse suplimentare despre utilizarea GitHub. Noi ne vom axa pe crearea unui repository. Pentru aceasta dați click pe + **New Repository**

Cont GitHub

The screenshot shows the GitHub homepage in a web browser. The browser's address bar displays "https://github.com". The GitHub logo is in the top left, and the user's profile name "demo-infoeducatie" is in the top right. Below the navigation bar, the "News Feed" tab is selected. The main content area features the "GitHub Bootcamp" guide, which consists of four steps: 1. Set up Git, 2. Create repositories, 3. Fork repositories, and 4. Work together. Below the bootcamp guide, there is a "Welcome to GitHub! What's next? (5 days ago)" section with links to "Create a repository", "Tell us about yourself", "Browse interesting repositories", and "Follow @github on Twitter". On the right side, the "Your repositories" section shows "0" repositories and a green "+ New repository" button, which is circled in red. Below this, a message states "You don't have any repositories yet! Create your first repository or learn more about Git and GitHub." At the bottom right, there is a link to "Subscribe to your news feed".

GitHub

Search GitHub

demo-infoeducatie

News Feed Pull Requests Issues

GitHub Bootcamp

- 1 Set up Git**
A quick guide to help you get started with Git.
- 2 Create repositories**
Repositories are where you'll work and collaborate on projects.
- 3 Fork repositories**
Forking creates a new, unique project from an existing one.
- 4 Work together**
Send pull requests, follow friends. Star and watch projects.

Welcome to GitHub! What's next? (5 days ago)


- Create a repository
- Tell us about yourself
- Browse interesting repositories
- Follow @github on Twitter

ProTip! Edit your feed by updating the users you follow and repositories you watch.

Your repositories 0 **+ New repository**

You don't have any repositories yet!
Create your first repository or learn more about Git and GitHub.

Subscribe to your news feed



Următorul pas este alegerea unui nume pentru repository. În câmpul „*Description*” puteți adăuga o scurtă descriere a proiectului. Nu uitați să bifați „**Initialize this repository with a README**”.

Opțional puteți alege, în partea de jos a paginii, adăugarea unui fișier .gitignore și/sau a unei licențe.


Fișierul .gitignore este folosit de git pentru a ignora fișierele pe care nu le doriți în repository, de exemplu: fișiere generate la compilare, fișiere private, etc.

Mai multe detalii puteți găsi aici: help.github.com.

Licența folosită determină condițiile în care o altă persoană poate folosi proiectul vostru. Un ghid alegerea unei licențe poate fi găsit aici: choosealicense.com.

Owner

Repository name

 demo-infoeducatie / MyFirstRepo ✓


Great repository names are short and memorable. Need inspiration? How about **tripping-octo-ironman**.

Description (optional)

Demo repository

☒  **Public**

Anyone can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.



Add .gitignore: **None**

Add a license: **None**



Create repository



După apăsarea butonului „**Create Repository**” veți ajunge pe pagina repository-ului nou creat.

The screenshot shows a web browser window displaying the GitHub interface for a newly created repository named 'MyFirstRepo' under the user 'demo-infoeducatie'. The browser's address bar shows the URL 'https://github.com/demo-infoeducatie/MyFirstRepo'. The repository page includes a header with the repository name and icons for watching (1), starring (0), and forking (0). Below this, a summary bar indicates '1 commit', '1 branch', '0 releases', and '1 contributor'. The main content area shows the 'Initial commit' by 'demo-infoeducatie' just now, with a 'README.md' file listed. The right sidebar contains links to 'Code', 'Issues' (0), 'Pull requests' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. At the bottom right, there are buttons for 'Clone in Desktop' and 'Download ZIP', along with the HTTPS clone URL 'https://github.com/...'.

demo-infoeducatie/MyFirstRepo

Unwatch 1 Star 0 Fork 0

Demo repository — Edit

1 commit 1 branch 0 releases 1 contributor

branch: master MyFirstRepo / +

Initial commit

demo-infoeducatie authored just now latest commit 849ef7660b

README.md Initial commit just now

MyFirstRepo

Demo repository

Code

Issues 0

Pull requests 0

Wiki

Pulse

Graphs

Settings

HTTPS clone URL

https://github.com/...

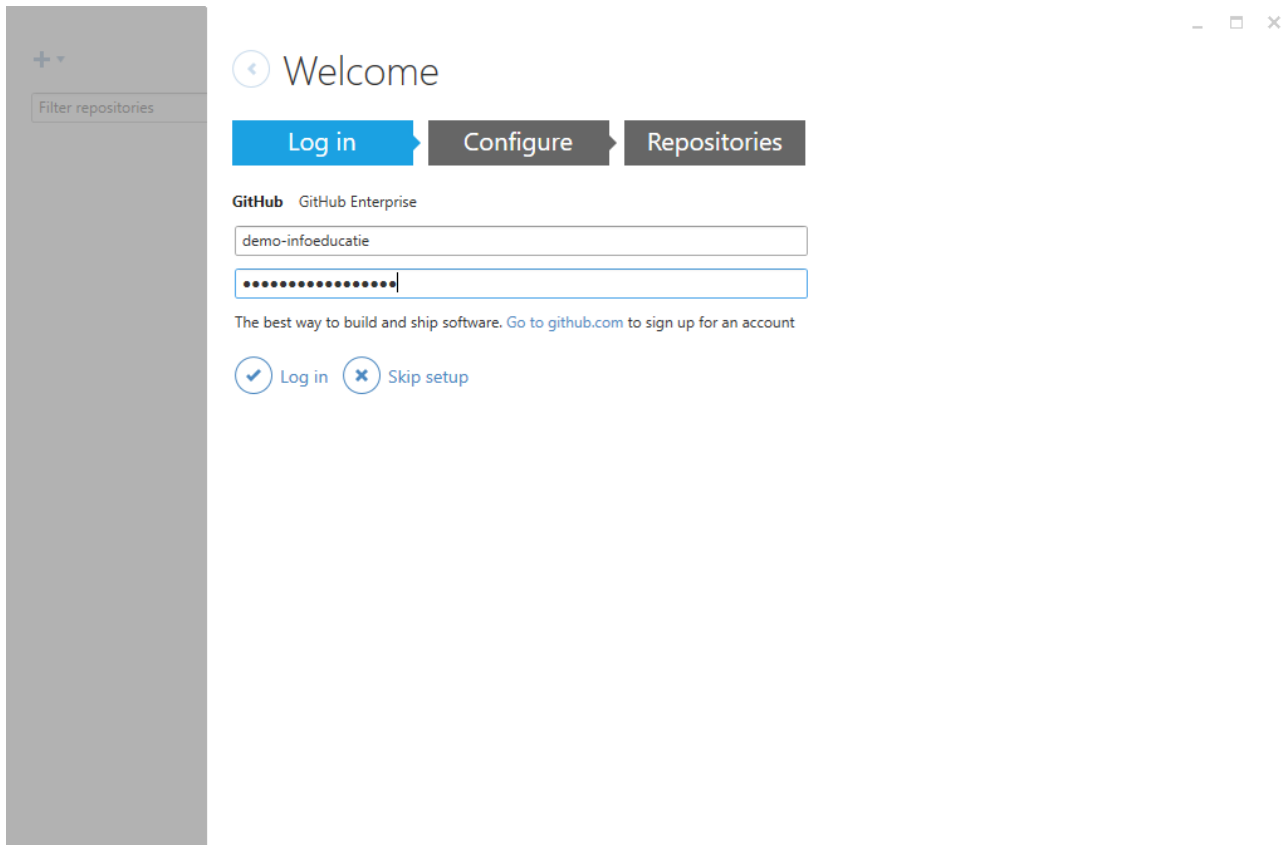
You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

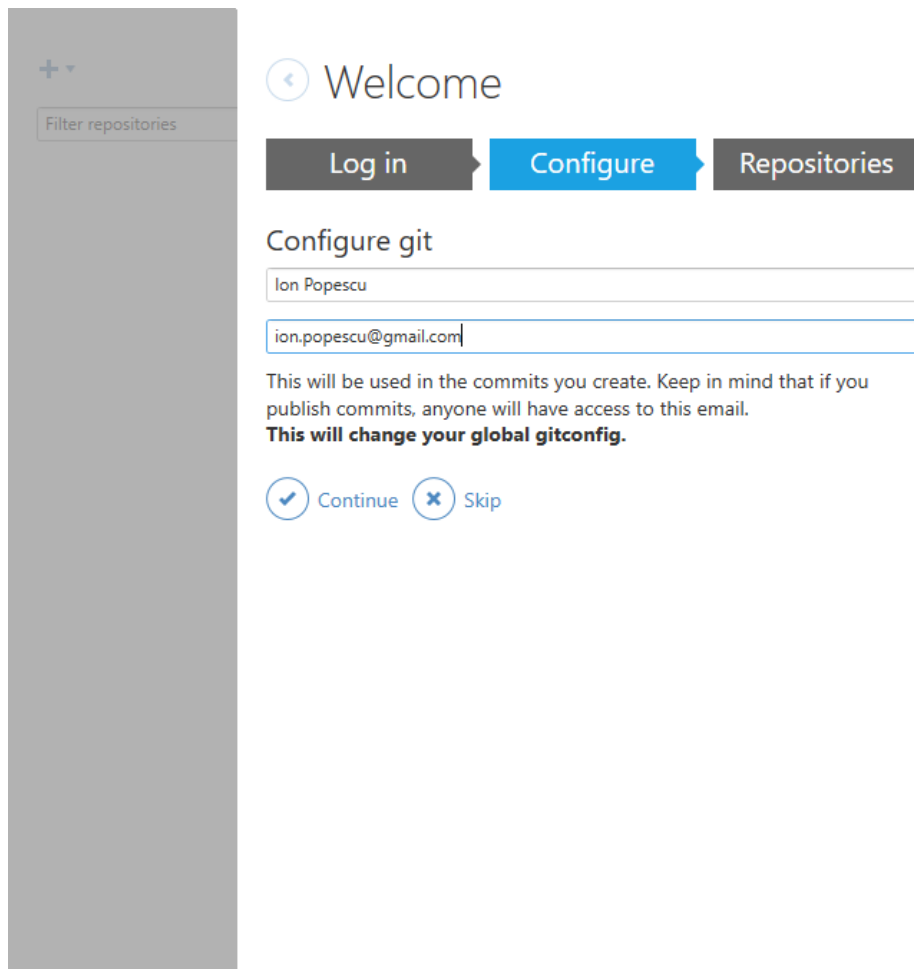
Download ZIP

3. Instalarea și configurarea GitHub for Windows

În continuare vom descăra și instala aplicația GitHub for Windows de la adresa windows.github.com. După instalare va trebui să vă autentificați în aplicație cu username-ul/adresa de mail și parola alese la crearea contului de GitHub.



În următorul pas va trebui să configurați identitatea voastră. Scrieți numele întreg și adresa de email. Acestea vor fi publice tuturor persoanelor care au acces la repository.



The screenshot shows the GitHub 'Configure git' interface. On the left is a sidebar with a search icon and a 'Filter repositories' input field. The main content area has a 'Welcome' header with a back arrow. Below it are three buttons: 'Log in', 'Configure' (highlighted in blue), and 'Repositories'. The 'Configure git' section contains two input fields: the first for the name 'Ion Popescu' and the second for the email 'ion.popescu@gmail.com'. A warning message states: 'This will be used in the commits you create. Keep in mind that if you publish commits, anyone will have access to this email. This will change your global gitconfig.' At the bottom are two buttons: 'Continue' (with a checkmark icon) and 'Skip' (with an 'x' icon).

Filter repositories

Welcome

Log in Configure Repositories

Configure git

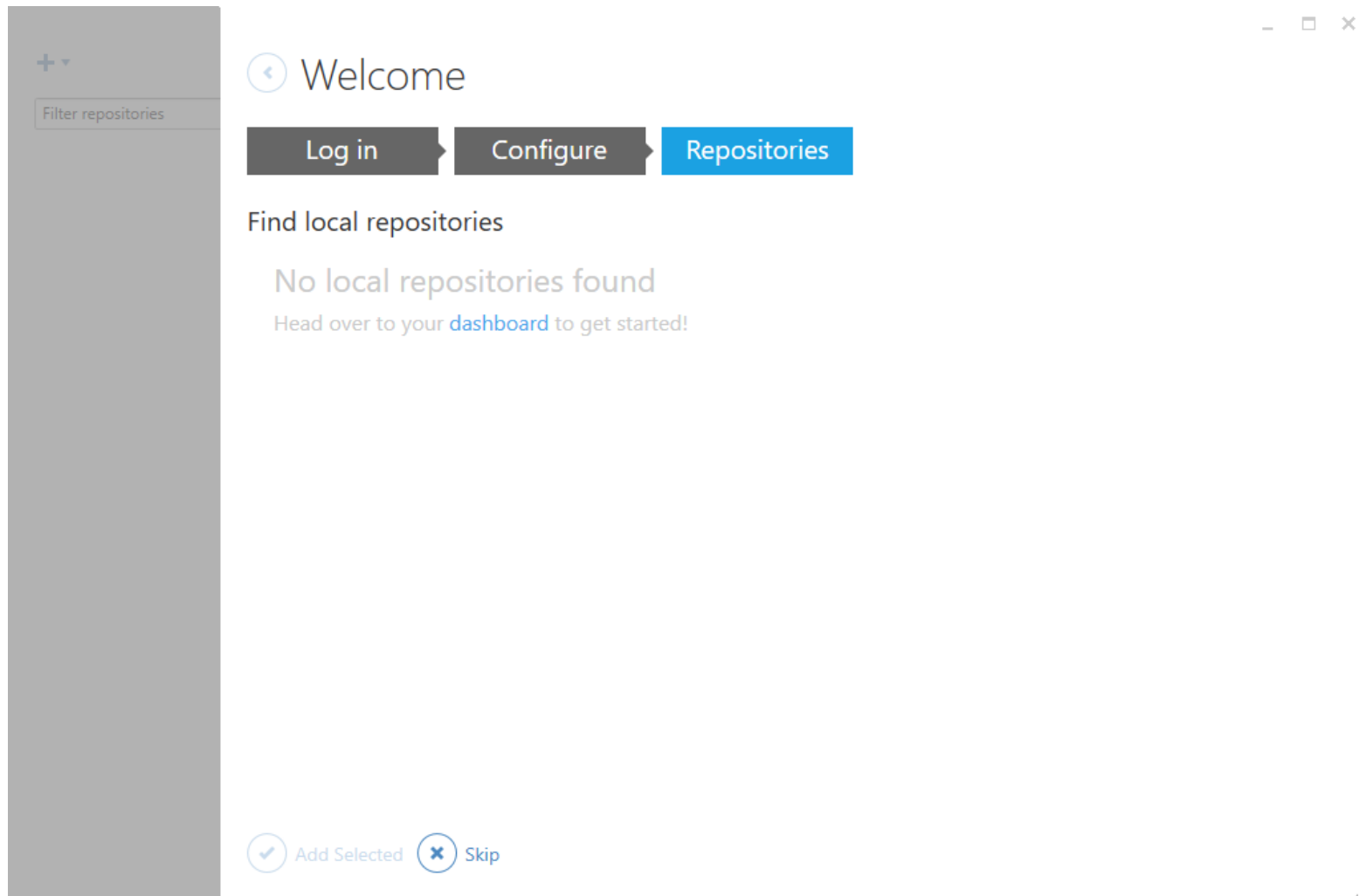
Ion Popescu

ion.popescu@gmail.com

This will be used in the commits you create. Keep in mind that if you publish commits, anyone will have access to this email.
This will change your global gitconfig.

Continue Skip

La ultimul pas puteți apăsa pe „**Skip**”, deoarece încă nu aveți niciun repository local.



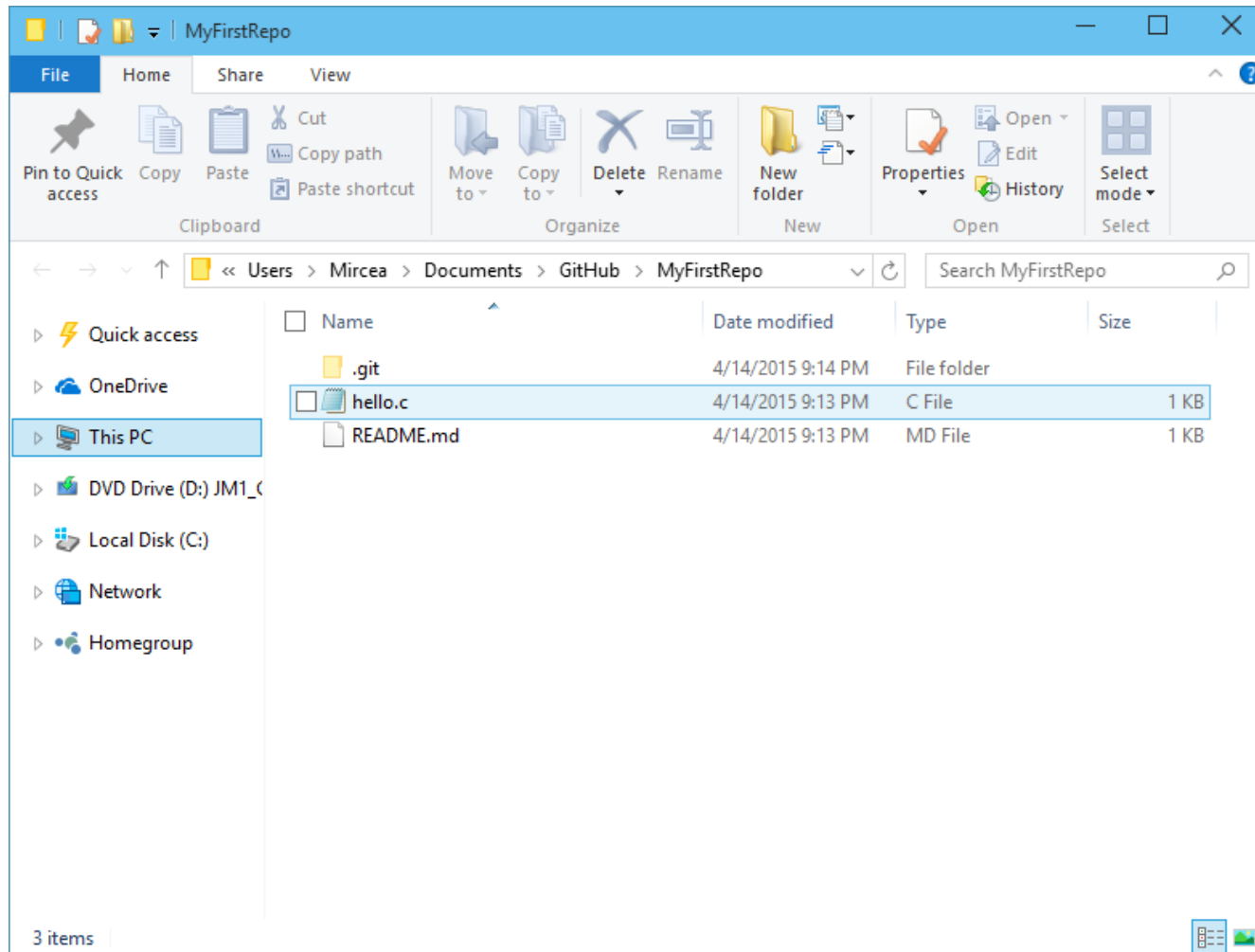
4. Crearea primului commit

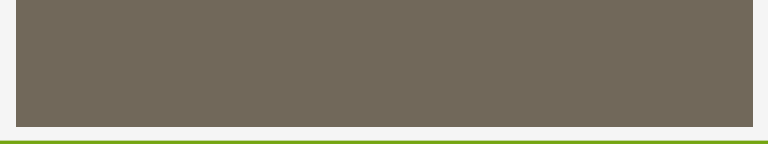
Acum vom clona (crea o copie locală) repository-ul creat anterior. Din aplicație vom da click pe „+”, vom selecta tab-ul „**Clone**”, contul și repository-ul pe care dorim să-l clonăm, iar în final vom apăsa „**Clone <repo-name>**”. Pe ecran va apărea o fereastră pentru selectarea directorului unde se va face clonarea.



adding a repository.

Acum puteți adăuga fișierele sursă în folderul în care ați clonat repository-ul. În acest exemplu este vorba de fișierul *hello.c*.





Reveniți în aplicația GitHub for Windows și selectați repository-ul. Veți observa pe coloana centrală că a detectat modificarea unor fișiere („*Uncommitted changes*”). Dați click pe Show pentru a vedea lista lor.

În coloana din dreapta vor apărea toate fișierele noi apărute în directorul repository-ului (ex: *hello.c*), precum și fișierele existente care au fost modificate de la ultimul commit (ex: *README.md*). Pentru a adăuga toate fișierele în repository bifați „*Files to commit*”.

Dacă doriți un control mai fin asupra fișierelor puteți expanda fiecare fișier și selecta doar anumite linii.

La final scrieți un mesaj de commit în câmpul Summary de pe coloana centrală și dați click pe commit.

+

▼

Filter repositories

MyFirstRepo

master ▼

Uncommitted changes

Hide ▲

Initial import **mesaj commit**

Import sources to repository

✓

Commit to master

2 files to be committed

History

Initial commit

53 minutes ago by demo-infoeducatie

Files to commit

Collapse all

hello.c

NEW

...

...

@@ -0,0 +1,7 @@

1 + #include <stdio.h>

2 +

3 + int main()

4 + {

5 + printf("Hello world\n");

6 + return 0;

7 + }

README.md

...

...

@@ -1,2 +1,4 @@

1 1 # MyFirstRepo

2 2 Demo repository

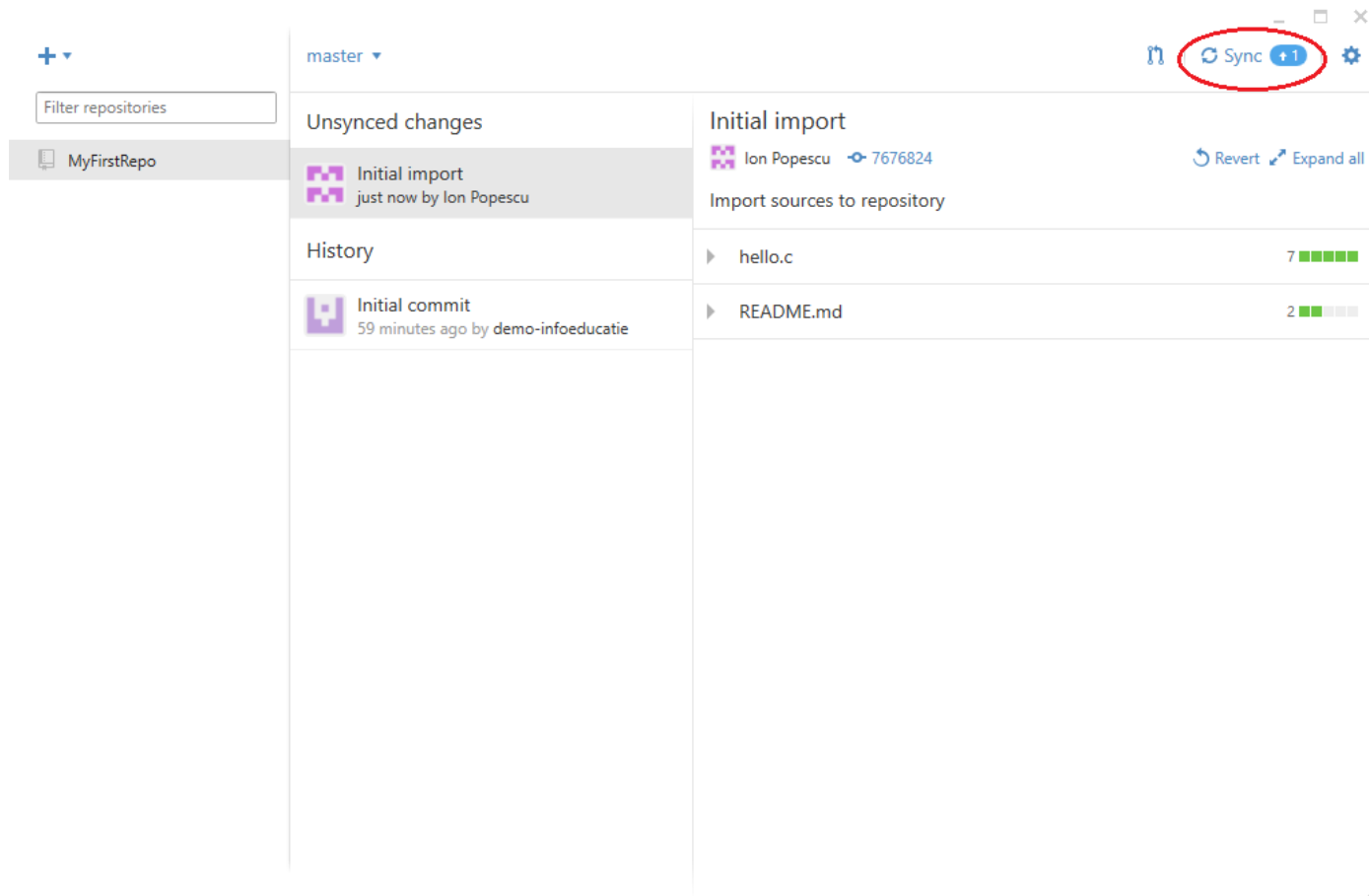
3 3 +

4 4 + Hello World!

3 5 \ No newline at end of file

Ultimul pas este sincronizarea repository-ului local cu GitHub. Tot ce trebuie făcut este să dați click pe „**Sync**”.

Important! După fiecare commit este necesară sincronizarea cu GitHub pentru ca modificările să fie disponibile și online.



Acum modificările create în ultimul commit vor fi vizibile tuturor persoanelor cu acces la repository.

The screenshot shows a web browser window displaying a GitHub repository page. The browser's address bar shows the URL <https://github.com/demo-infoeducatie/MyFirstRepo>. The repository page header includes the GitHub logo, a search bar, and navigation links: Explore, Gist, Blog, and Help. The repository name 'demo-infoeducatie / MyFirstRepo' is prominently displayed, along with interaction buttons: 'Unwatch' (1), 'Star' (0), and 'Fork' (0). Below the header, the repository is identified as a 'Demo repository' with an 'Edit' link. A summary bar shows '2 commits', '1 branch', '0 releases', and '1 contributor'. The main content area shows the 'branch: master' selected, with a '+ MyFirstRepo' button. A list of files is shown, including 'README.md' and 'hello.c', both marked as 'Initial import' and dated '4 minutes ago'. The 'README.md' file content is visible, showing the title 'MyFirstRepo', the subtitle 'Demo repository', and the text 'Hello World!'. On the right side, a sidebar contains links to 'Code', 'Issues' (0), 'Pull requests' (0), 'Wiki', 'Pulse', 'Graphs', and 'Settings'. At the bottom of the sidebar, the 'HTTPS clone URL' is provided as <https://github.com/>, with a note that the repository can also be cloned using HTTPS, SSH, or Subversion. Two buttons are present at the bottom: 'Clone in Desktop' and 'Download ZIP'.

Bibliografie

- <https://git-scm.com>
- <https://git-scm.com/book/ru/v1/Введение-Основы-Git>
- <https://githowto.com/ru>
- <http://eax.me/git-commands/>
- <https://habrahabr.ru/post/174467/>
- <http://rogerdudler.github.io/git-guide/>
- <https://proglib.io/p/git-for-half-an-hour/>