# Tema: Çfarë është Git?

# Lista e figurave

Përshëndetje të gjithëve!

Para disa kohësh ju kam pyetur se sa prej jush e dinin çfarë është **GitHub**. Pas rezultatit të pyetësorit mendova se ishte e përshtatshme një postim mbi këtë temë. Por, para se të flasim për GitHub duhet të kuptojmë çfarë është *Version Control* dhe çfarë është *Git*.

## Çfarë është Version Control?

**Version Control (VC)** ndihmon zhvilluesit të gjurmojnë dhe menaxhojnë ndryshimet në kodin e një projekti softuerik. Gjatë kohës që një projekt softuer zhvillohet, kontrolli i versionit bëhet thelbësor.
**Version control systems (VCS)** janë mjete softuerike të cilat ndihmojnë skuadrat e zhvilluesve të menaxhojnë ndryshimet në *source code* gjatë zhvillimit. *Shembull*: Kur zhvilluesit krijojnë diçka, si për shembull aplikacion, bëjnë ndryshime të vazhdueshme të kodit, duke nxjerrë versione të reja. **VCS** i mban këto rishikime saktë, duke i ruajtur modifikimet në një **repository**[1] qendrore. Kjo i lejon zhvilluesit të bashkëpunojnë lehtësisht, pasi ata mund të shkarkojnë një version të ri të softuerit, të bëjnë ndryshime dhe të ruajnë ndryshimet e reja. Më pas çdo zhvillues mund t'i shohë këto ndryshime të reja, t'i shkarkojë dhe të punojë mbi to. Kjo bën të mundur që çdo zhvillues mund të bëjë ndryshime në çdo pjesë të projektit dhe më pas t'i ruajë. VC ndihmon me këtë proces pasi gjurmon çdo ndryshim të kryer në një projekt.

**Pse Version Control është i rëndësishëm?**
- Përmirëson vizibilitetin
- Ndihmon ekipet të bashkëveprojnë në gjithë botën
- Përshpejton dorëzimin e produktit
- Mundëson gjurmueshmërinë për çdo ndryshim të bërë
- Përshpejton kërkimin e file-ve

**Tipet e *Version control systems*:**

- ***VCS lokal (local):*** është një bazë të dhënash lokale e vendosur në kompjuterin tuaj.

- ***VCS i centralizuar (centralized)***: të gjitha ndryshimet në file gjurmohen nën serverin e centralizuar. Keni një kopje të vetme "qendrore" të projektit tuaj në një server dhe bëni ndryshimet në këtë kopje qendrore. Mund të merrni file-t që ju nevojiten, por kurrë nuk keni një kopje të plotë të projektit tuaj në nivel lokal. *Shembull:* Subversion (SVN) dhe Perforce.

- ***VCS i shpërndarë (distributed)***: nuk mbështeteni në një server qendror për të ruajtur të gjitha versionet e file-ve të një projekti si VCS i centralizuar. Në vend të kësaj, klononi një

[1] një strukturë të dhënash që ruan metadata për një grup file-sh ose directory structure

kopje të një *repository* lokalisht, në mënyrë që të keni historinë e plotë të projektit në kompjuterin tuaj. ***Shembull:*** Git dhe Mercurial.

## Çfarë është Git?

**Git** është një *VCS i shpërndarë* open-source, i krijuar për të trajtuar nga projekte të vogla në projekte të mëdha, me shpejtësi dhe efikasitet. Është krijuar nga Linus Torvalds - *i njëjti person që ka krijuar Linux,* në 2005. Sipas *Stack Overflow developer survey*, mbi 93.43% e zhvilluesve përdorin Git.

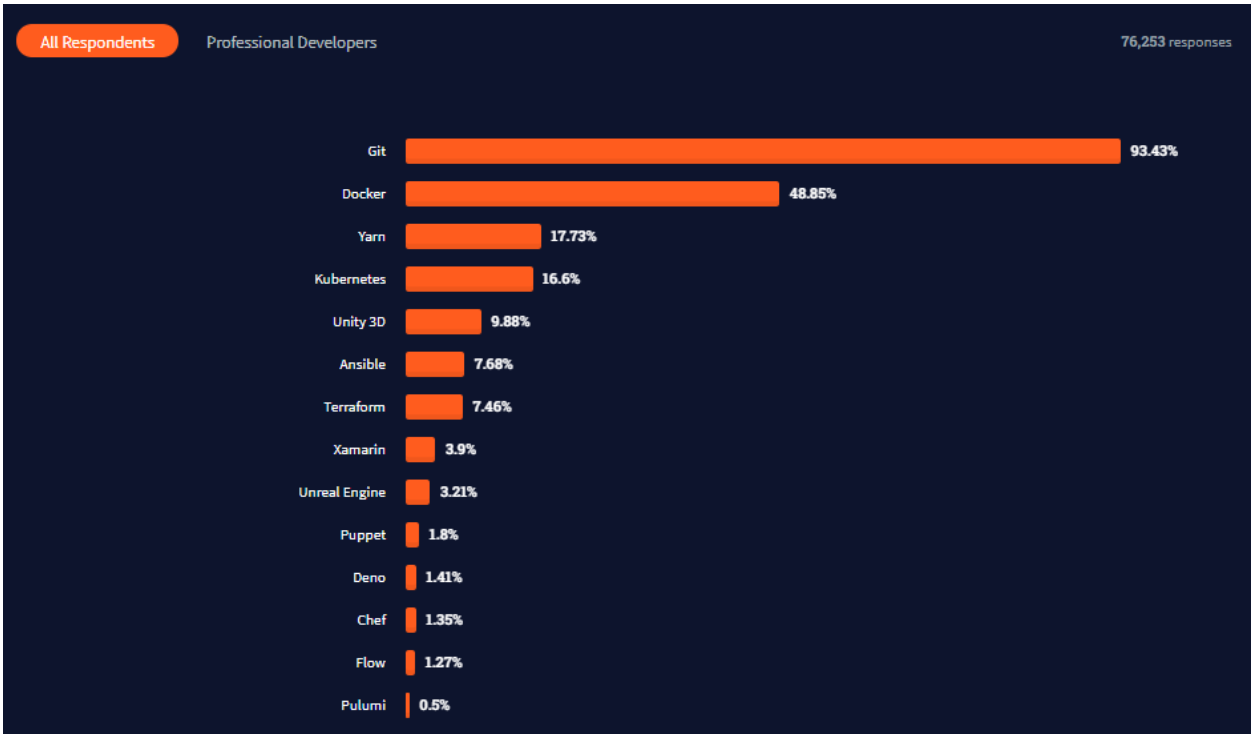


*Fig. 1 Stack Overflow developer survey*
Source

**Git** është një *VCS i shpërndarë* që gjurmon ndryshimet në file-t e kompjuterit dhe përdoret zakonisht në zhvillime softuerike për të mbajtur ndryshimet midis file-ve të ndryshëm. Në rast se ndryshimet e reja prishin diçka, Git mund të përdoret për të hyrë në file-t specifike para se të bëheshin ndryshimet dhe mund të rivendosen file-t e mëparshëm derisa ndryshimet të rregullohen. Gjithashtu ruan se kush ka kryer cilin modifikim. Git mund të instalohet në sistemet Windows, Linux dhe MacOS (instalo).

**Fjalor që lidhet me VCS:**

**Repository:** mund të përshkruhet si "zemra" e çdo VCS. Repository është vendi qendror ku të gjithë programuesit punojnë dhe ruajnë kodin e tyre. Përveç ruajtjes së file-ve, repository ruan gjithashtu historinë e ndryshimeve.

**Tags:** është një emër (etiketë) i lidhur me një *commit* specifik. Ato ndihmojnë në krijimin e *snapshots* të projektit. Krijimi i *tags* na lejon të mbajmë emra përshkrues për një version specifik të projektit në *repository* të cilit mund t'i rikthehemi.

**Branch**: mund ta imagjinojmë si degët e pemës. Është një version i *repository*, ose me fjalë të tjera, një linjë e pavarur zhvillimi. Një *repository* mund të përmbajë *branches* të shumta, që do të thotë se ka shumë versione të tij. Gjatë zhvillimit programuesi krijon një branch për çdo task të ri. Pasi mbaron ndryshimet, ky *branch* bëhet *merge* me branch-in kryesor (zakonisht quhet master) ku ruhen ndryshimet përfundimtare.

**Working copy:** është një snapshot e *repository* ku zhvilluesi po punon. Secili zhvillues ka një *working copy* lokalisht. Ndryshimet e bëra në këtë "kopje" bëhen merge në *repository*-n kryesor. Ndryshe mund të konsiderohet si një vend pune privat ku zhvilluesit mbajnë punën e tyre, e cila është e izoluar nga pjesa tjetër e zhvilluesve në projekt.

**Commit changes:** është procesi i ruajtjes së ndryshimeve nga *working copy* në serverin qendror. Pas një *commit*-i të suksesshëm, çdo zhvillues tjetër në projekt mund të shohë ndryshimet e bëra si dhe mund t'i marrë dhe të punojë mbi to në lokal.

**Komandat kryesore në Git:**

- **git init:** përdoret nëse duam të krijojmë një *repository* bosh ose të ri-inicializojmë një ekzistues. Do të krijohet një *directory* **.git**.

  *Komanda:* `git init <repository name>`

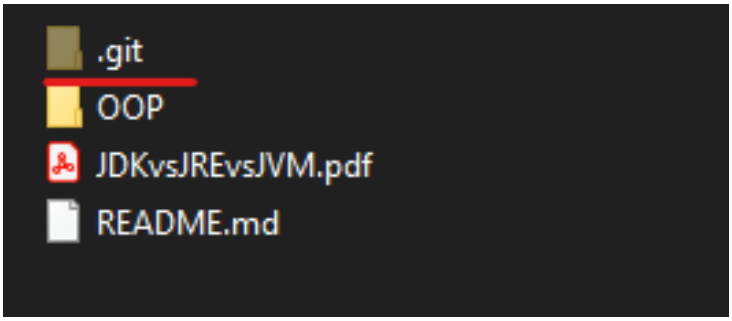


*Fig. 2 Directory .git*

- **git clone:** përdoret për të shkarkuar versionin më të fundit të një projekti në remote dhe për ta kopjuar në makinën lokale.

  *Komanda:* `git clone <repository url>`

- **git fetch:** merr të gjitha azhornimet nga *remote repository*, përfshirë *branch*-et e reja.

  *Komanda*: `git fetch`

- **git branch**: përdoret për krijimin, listimin dhe fshirjen e *branches*.

  *Komanda:*

  - Krijimi i një *branch* të ri: **`git branch <branch-name>`**
  - Listimi i *branch*-ve ekzistues: **`git branch / git branch –list`**
  - Fshirja e një branch: **`git branch -d <branch name>`**

- **git checkout:** për të punuar në një *branch*, së pari duhet të kalojmë te ai *branch*. Git checkout përdoret kryesisht për kalimin nga një branch në tjetrin.

  *Komanda:* **`git checkout <branch name>`**

  *Për të ndryshuar branch duhen plotësuar disa kushte:*

  1. Ndryshimet në *branch*-in aktual duhet të bëhen ***commit*** ose ***stash*** para se të ndërrojmë branch
  2. Branch-i në të cilin duam të kalojmë, duhet të ekzistojë në versionin local

  Ekziston dhe një komandë e cila na lejon ta krijojmë dhe të kalojmë direkt në *branch*-in e Krijuar. *Komanda:* **`git checkout -b <branch name>`**

- **git status:** jep të gjithë informacionin e nevojshëm në lidhje me *branch*-in aktual.

  *Komanda:* git status

  *Një file ka disa gjendje (states):*

  **Untracked:** nuk është një gjendje e vërtetë e Git. File-t quhen *untracked* kur ato nuk ekzitojnë në snapshotin (commit) e fundit. Sa herë që një file i ri shtohet në *working directory* që nuk ka ekzistuar më parë, konsiderohet si një *untracked file*. Kjo ndodh, sepse Git e sheh si një file që nuk ka ekzistuar në snapshotin (commit) e mëparshëm. Marrim një shembull. Krijojmë një *branch* të ri në JavaInDetail, të quajtur **test-branch** dhe kalojmë në këtë *branch*. Nëse shohim **statusin** e këtij branch-i të sapo krijuar kemi mesazhin ***"nothing to commit, working tree clean"***. Kjo tregon që akoma nuk kemi kryer ndryshime

në këtë branch. Më pas krijojmë një file **'testfile.txt'**. Nëse do të shohim përsëri status me komandën **git status** shohim se te kategoria **Untracked files** ndodhet file i ri që krijuam.

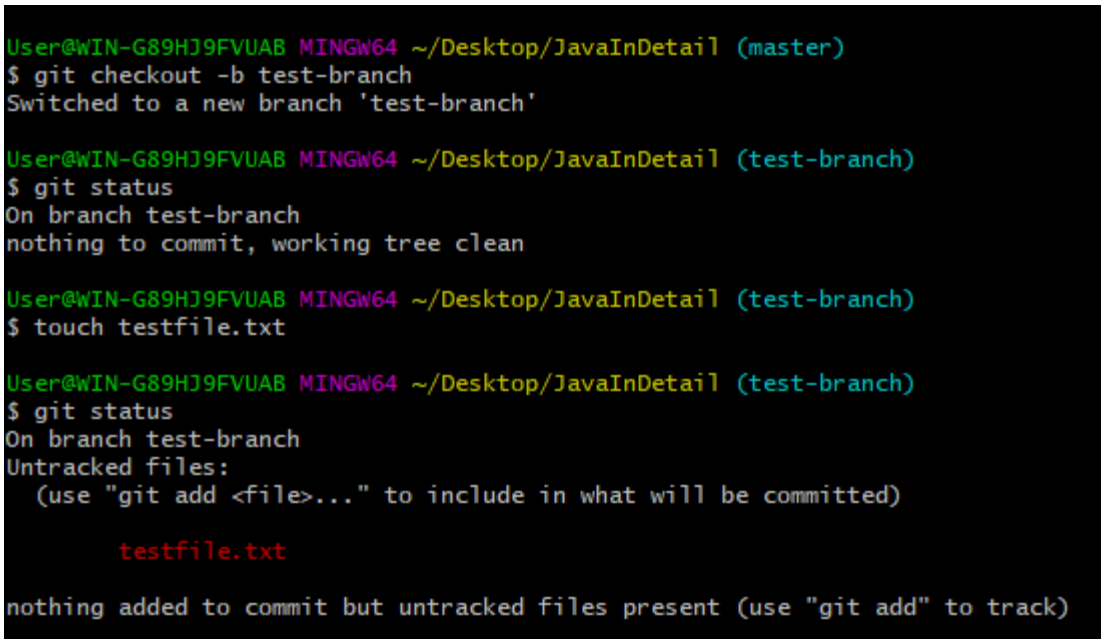


*Fig. 3 Git Untracked files*

**Modified:** do të thotë që file është ndryshuar, por nuk është bërë commit akoma në database-n e git. Nëse file është ekzistues dhe bëjmë një ndryshim në të, atëherë statusi do të tregojë **modified: <filename>** si më poshtë:

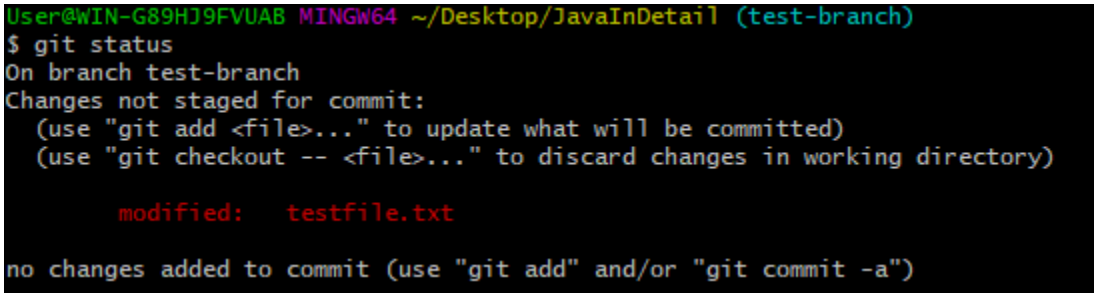


*Fig. 4 Git Modified files*

**Staged:** do të thotë se file nuk është i pranishëm në *commit*-in e fundit (psh. file-t e sapo krijuara) ose është një file i modifikuar që zhvilluesi kërkon ta përfshijë në *commit*-in e radhës. Pra, në *gjendjen staged* kalohen file-t të cilat duam të bëjmë *commit*. Filet shtohen në **gjendjen staged** duke përdorur komandën **git add**. Dy lloje file-sh mund të shtohen në një gjendje staged: *untracked* ose *të modifikuara*.

```
User@WIN-G89HJ9FVUAB MINGW64 ~/Desktop/JavaInDetail (test-branch)
$ git add  testfile.txt

User@WIN-G89HJ9FVUAB MINGW64 ~/Desktop/JavaInDetail (test-branch)
$ git status
On branch test-branch
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   testfile.txt
```

*Fig. 5 Git Staged files*

**Committed:** pasi të bëhen ndryshimet në lokal, mund t'i ruajmë duke i bërë "commit". Një commit është si të vendosësh një pikë kontrolli në procesin e zhvillimit të cilës mund t'i kthehesh më vonë nëse është e nevojshme. Gjithashtu gjatë *commit* duhet të shkruajmë një mesazh të shkurtër për të shpjeguar atë që kemi zhvilluar ose ndryshuar në source code në mënyrë që të dokumentojmë ndryshimet.

```
User@WIN-G89HJ9FVUAB MINGW64 ~/Desktop/JavaInDetail (test-branch)
$ git commit -m "Ndryshim ne testfile.txt"
[test-branch bdd3bdf] Ndryshim ne testfile.txt
 1 file changed, 1 insertion(+)
```

*Fig. 6 Git Commit files*

**E RËNDËSISHME!**
***Git commit* ruan ndryshimet vetëm lokalisht.**

- **git add:** përdoret për të shtuar file-t në *fazën staged*, pra përzgjedhim cilat file duam të bëjmë commit.
  *Komanda:* **`git add <file>`**
  Nëse duam të shtojmë *të gjitha* file-t e modifikuara në *fazën staged* atëherë mund të përdorim komandën: **`git add -A`** ose **`git add .`**

- **git commit:** (shpjegimi në gjendjen committed më sipër)

- **git push:** pas kryerjes së ndryshimeve lokalisht, gjëja tjetër që duam të bëjmë është të "dërgojmë" ndryshimet lokale në serverin remote.

  Nëse branch-i **ekziston** në remote mund të përdorim komandën: **`git push`**

**E RËNDËSISHME!**
***Git push* ngarkon në *remote repository* vetëm ndryshimet e bëra *commit*.**

Nëse branch-i **nuk ekziston** në remote përdorim komandën: **`git push --set-upstream <remote branch> <branch name>`** ose **`git push -u origin <branch name>`**

- **git pull:** përdoret për të marrë përditësimet nga *remote repository*. Është një kombinim i dy komandave ***git fetch*** dhe ***git merge*** që do të thotë se, ***git pull*** merr përditësimet nga *remote repository* (git fetch) dhe aplikon menjëherë ndryshimet më të fundit në mjedisin tonë lokal (git merge).
  *Komanda:* **`git pull <remote branch>`**

- **git diff:** tregon ndryshimet midis commits, commit dhe ndryshimeve në lokal etj.

```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   testfile.txt


User@WIN-G89HJ9FVUAB MINGW64 ~/Desktop/JavaInDetail (test-branch)
$ git diff testfile.txt
diff --git a/testfile.txt b/testfile.txt
index b0922a9..3b12464 100644
--- a/testfile.txt
+++ b/testfile.txt
@@ -1 +1 @@
-sffffffff
\ No newline at end of file
+TEST
\ No newline at end of file
```

*Fig. 7 Git diff*

Bëjmë një ndryshim në file-n *testfile.txt* dhe me anë të komandës **`git diff testfile.txt`** shohim që është fshirë teksti i mëparshëm 'sffffffff' dhe është shtuar 'TEST'.

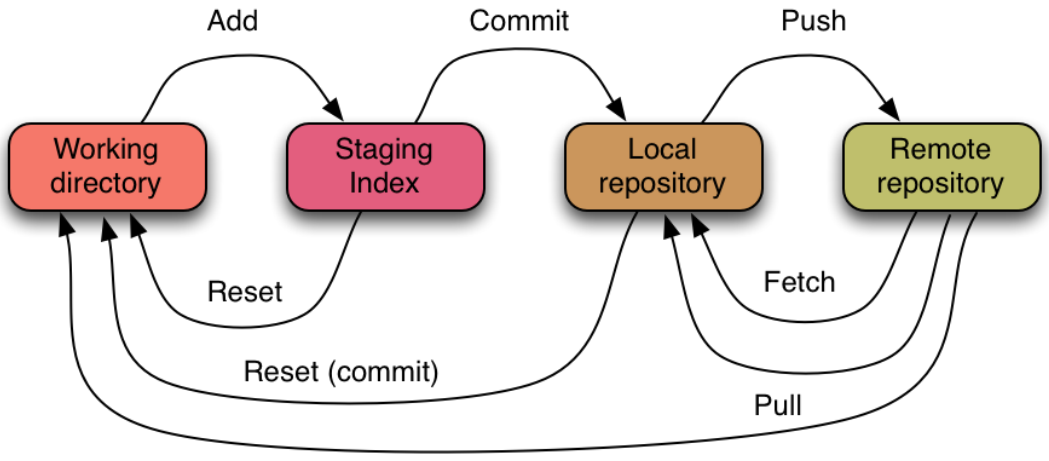


*Fig. 8 Git*

Source

## Përfundim

Në këtë shkrim mësuam se çfarë është *Version Control* dhe si funksionojnë *Version Control Systems*. Ku nga të fundit më i përdoruri është *Git*. Gjithashtu, mësuam dhe disa prej komandave kryesore të git të cilat do t'i përdorni shpesh gjatë punës suaj.

***Lexime të mëtejshme:***

https://git-scm.com/docs

https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet