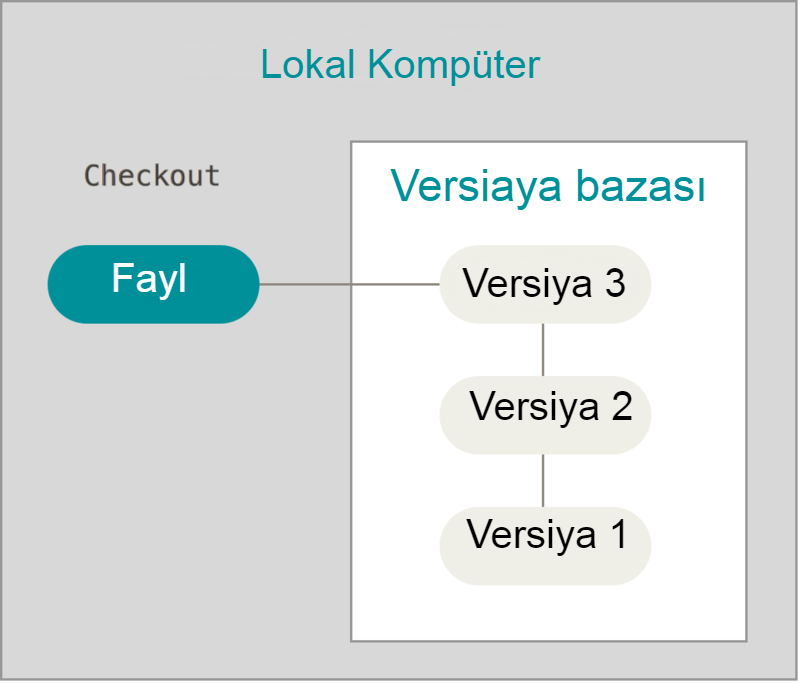
Git Və GitHub. Ən Çox İstifadə Olunan Əmrlər (Part 1).

[**Git**](https://git-scm.com/) bir versiya kontrol sistemidir (Version Control System — VCS). Version control dedikdə nə nəzərdə tutulur? Biz kod yazmağa başladıqdan etibarən ilk faylımızı yaratmaqla onun ilk versiyasını yaratmış oluruq. Və hər dəfəsində onun üzərində dəyişiklik edib yaddaşda saxladıqca onun yeni versiyalarını yaradırıq. Bir fayl üzərində edilən dəyişikliklər yaddaşa qeyd edilmədiyi müddətdə həmin faylın yeni versiyası yaranmış sayılmır. Version control biz fayl üzərində hər dəyişiklik etdikdən sonra onun dəyişiklikdən əvvəlki halı və ya halları ilə sonrakı halı arasındakı fərqləri ortaya çıxarır.

Proqramlaşdırmada versiya nəzarətini həyata keçirmək üçün bir çox versiya nəzarət sistemləri mövcuddur:

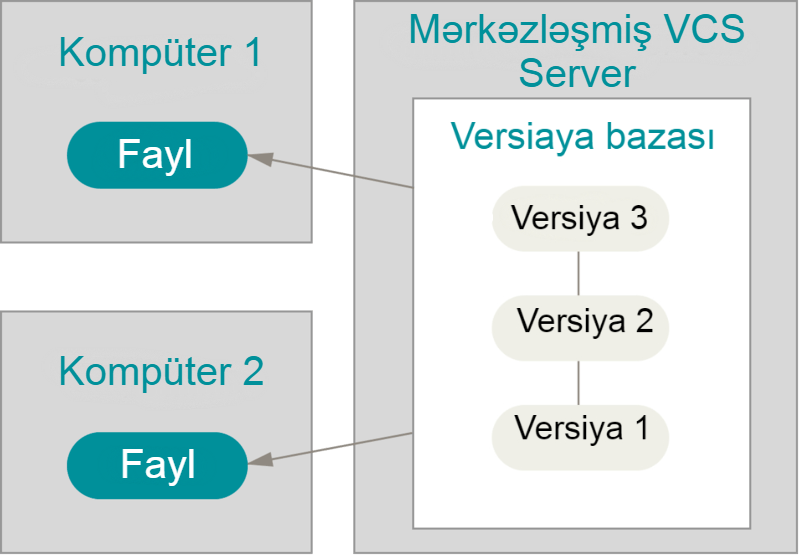
* **Subversion**
* **Git**
* **CVS**
* **Mercurial**
* **ClearCase**
* **Team**
* **Foundation**
* **Visual SourceSafe**

Versiya kontrol sistemlərinin lokal (local), mərkəzləşmiş (central) və paylanmış (distributed) arxitekturaya malik modelləri mövcuddur.



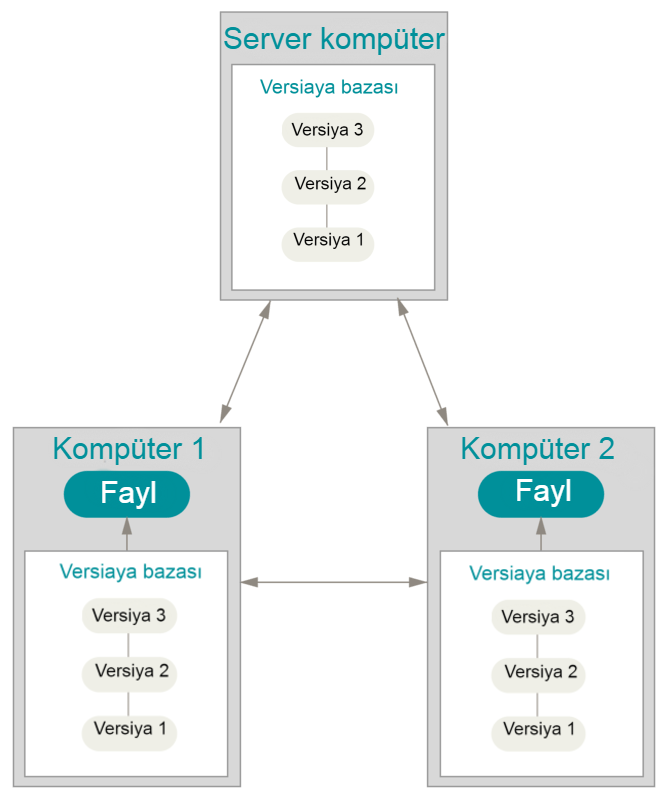
Lokal arxitekturalı VCS

**Lokal VCS** lokal sistem daxilində faylları izləyir. Bu yanaşma çox yayğın və sadədir. Bu VCS səhvlərə meyllidir, yəni səhv fayla təsadüfən yazma şansının daha yüksək olduğu deməkdir.



Mərkəzləşmiş arxitekturalı VCS

Bu yanaşmada fayllardakı bütün dəyişikliklər **mərkəzləşdirilmiş server** altında izlənilir. Mərkəzləşdirilmiş serverə versiyalı faylların bütün məlumatları və həmin mərkəzdən faylları yoxlayan müştərilərin siyahısı daxildir.



Paylanmış arxitekturalı VCS

Git bu günə qədər dünyada ən çox istifadə olunan müasir versiya nəzarət sistemidir. Git 2005-ci ildə [Linux](https://az.wikipedia.org/wiki/Linux) əməliyyat sistemi [kernel](https://www.webopedia.com/TERM/K/kernel.html)inin məşhur yaradıcısı Linus Torvalds tərəfindən hazırlanmış və fəal şəkildə dəstəklənən açıq qaynaqlı layihədir. Gitin CVS və Subversion kimi digər məşhur versiya nəzarət sistemlərindən əsas fərqi **paylanmış versiya nəzarət sistemi** ([Distribution Version Control System — DVCS](https://www.perforce.com/blog/vcs/what-dvcs-anyway)) olmasıdır. Belə ki, subversion tək mərkəzli versiya nəzarət sistemi olmaqla bütün datanı bir mərkəzdə toplayır və versiyalayır. Git isə DVCS olaraq hər bir proqramçı üçün proqramın bütün tarixi versiya dəyişikliklərini özündə ehtiva edən bir baza kopyasını verir və hər bir proqramçı fərdi şəkildə onun üzərindən iş görür.

Bəs gitin hansı xüsusiyyətləri var?

**1. Qeyri-xətti inkişaf üçün güclü dəstək.**

Qeyri-xətti inkişaf bizə bir sistemin müxtəlif hissələrində eyni vaxtda işləməyə imkan verir. Beləliklə, müəyyən bir qaydada bir tətbiq qurmaq məcburiyyətində deyilsiniz. Əvvəl yazdığınız kodda bir səhv xatırladığınız zaman hal-hazırda işlədiyinizi saxlaya, əvvəlki işinizə asanlıqla qayıdıb səhvi düzəldə bilərsiniz.

**2. Paylanmış axitektura.**

Paylanmış axitektura çox insanın eyni layihə üzərində eyni vaxtda işləməsinə imkan verir. Versiya nəzarəti olmadan başqaları ilə eyni sənəddə işləyə bilməzsiniz, çünki dəyişikliklərdən başqasının xəbəri olmadığı üçün eyni kodun üzərinə yazacaqdır.

**3. Təhlükəsizlik tədbirləri.**

Git-də yaddaşda qeyd olunan hər şeyə sonradan baxıla bilər. **Hətta silsən də.**Tarixi dəyişikliklərin həmişə bir qeydi qalır. Beləliklə, təsadüfən və ya bilərəkdən kodunuzu silmək çətindir.

Git arxitekturasından istifadə edən bir neçə serverlər mövcuddur. Bunlardan ən məşhurları [GitHub](https://github.com/), [GitLab](https://about.gitlab.com/), [Bitbucket](https://bitbucket.org/)-dir. Git texnologiyalarından istifadə edə bilmək üçün ik növbədə [git](https://git-scm.com/)i öz kompüterimizə qurmalıyıq. Bundan sonra hər hansı bir git serverində yazdığımız layihələri saxlaya biləcəyimiz özümüzə aid bir hesab açmalıyıq (bizim nümunəmizdə bu GitHub hesabıdır). Kompüterimizə qurduğumuz gitin “GitBash” proqram pəncərəsini açmaqla öz GitHub hesabımızı lokalımızdakı git bazası ilə əlaqələndiririk. Bunun üçün siz GitHub`da qeydiyyatdan keçdiyiniz emaildan istifadə etməklə GitBash pəncərəsində aşağıdakı əmrləri daxil etməlisiniz.

git config --global user.name "Ad Soyad"  
git config --global user.email sizinemail@example.com

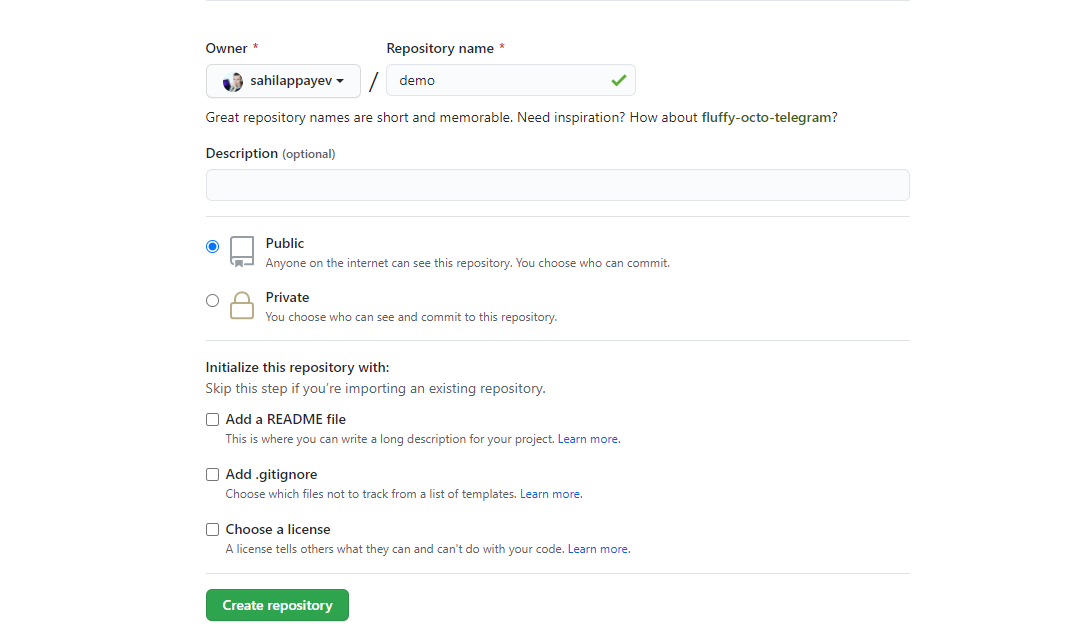
Ardından siz doğrulamaq şərti ilə iki dəfə istifadəçi şifrənizi daxil edəcəksiniz (şifrə daxil edilərkən ekranda görünməyəcək). Bu mərhələdən sonra artıq siz GitHub serverindəki hesabınızı öz kompüterinizdəki git bazasına bağlamış olursunuz.

Gitlə bağlı daha nələ edə bilərik bunu [növbəti](https://medium.com/pragmatech/git-v%C9%99-github-%C9%99n-%C3%A7ox-i%CC%87stifad%C9%99-olunan-%C9%99mrl%C9%99r-part-2-3cfbaa503564) məqaləmizdə əla almağa çalışacağıq.

# Git Və GitHub. Ən Çox İstifadə Olunan Əmrlər (Part 2).

[Əvvəlki](https://medium.com/pragmatech/git-v%C9%99-github-%C9%99n-%C3%A7ox-i%CC%87stifad%C9%99-olunan-%C9%99mrl%C9%99r-part-1-40d386679759) məqaləmizdə versiya nəzarət sistemləri (VCS), giti digər VCS-dən fərqləndirən əsas xüsusiyyətlər və giti öz kompüterimizə qurub git server (GitHub) hesabımızla necə əlaqələndirəcəyimizdən danışmışdıq. Bu məqaləmizdə gitin iş prisipi və başlanğıc səviyyədə bilməli olduğumuz git əmrlərini ələ almağa çalışacağıq.

Giti kompüterimizdə qurub GitHub hesabımızla əlaqələndirdikdən sonra gəlin ilk remote [repository](https://docs.github.com/en/github/creating-cloning-and-archiving-repositories/about-repositories)-imizi (repo) yaradaq. Bunun üçün biz GitHub hesabımıza daxil olub əsas səhifədə sol küncdə “Repositories” bölməsindən “New” düyməsinə sıxmaqla yeni repository yaratmaq üçün müvafiq səhifəni açırıq.

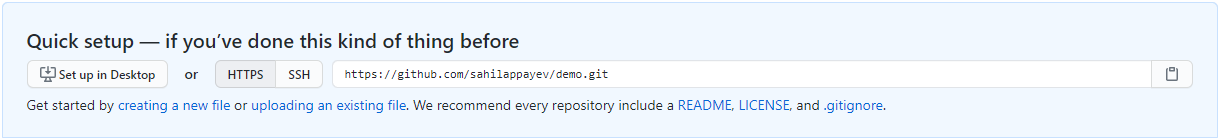


Burada yaradacağımız reponun adını (bizim nümunəmizdə: demo), proyektin hamıya açıq (public) və ya bağlı (private) olmağını təyin edə, proyekt haqqında məlumatları əks etdirən “README” faylını, proyektimizi repo-ya yüklədiyimiz zamanı yüklənməsini istmədiyimiz faylları qeyd etmək üçün “.gitignore” faylını və lisenziya sənədi əlavə edə bilərik.

LICENSE sənədi repoda saxladığınız sənədləri qoruyan lisenziyanızı müəyyənləşdirir. Onun şərtləri seçdiyiniz lisenziyadan asılı olaraq dəyişir. [Bu ünvanda](https://choosealicense.com/) 30 pulsuz lisenziyaya baxa bilərsiniz.

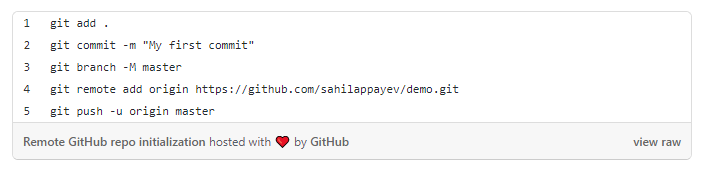
GitHub repo yaratdığınız zaman bu lisenziyalardan birini seçməyə imkan verir və avtomatik olaraq bu lisenziyanı repoya əlavə edir. Hər kəs lisenziya icazə verdiyi və lisenziyada göstərilən şərtlərə əməl edildiyi müddətdə lisenziyadan istifadə edə bilər. Pulsuz lisenziyaların mənası buradan qaynaqlanır. Lisenziyalar pulsuzdur ki, qoruduqları şeylər pulsuz olsun.

Description bölməsində isə proyektiniz haqqında qısa açıqlama yaza bilərsiniz.



Bundan sonra “Quick setup” bölməsində repomuzun GitHub ünvanını kopyalayır, kompüterimizdə proyektimiz olan qovluqda kontekst menyudan “Git Bash Here” düyməsini sıxmaqla git bash terminalını açırıq və lokal repomuzu yaratmaq üçün **git init**əmrini daxil edirik. Nəticədə gizli “.git” qovluğu yaranacaq ki, bu qovluq bizim gitlə bağlı bütün konfiqurasiya fayllarını və lokal repomuzu özündə saxlayır.

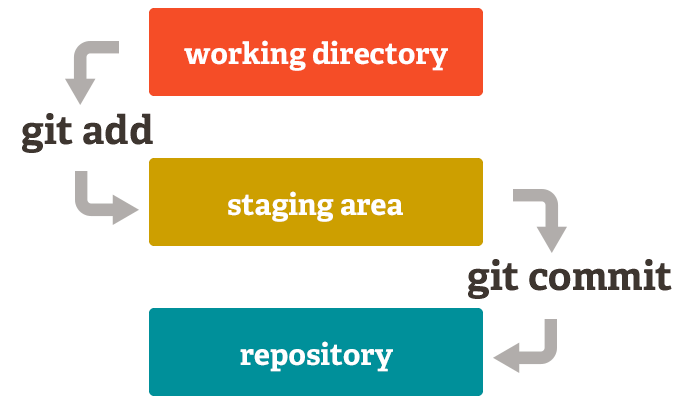
İndi isə yaratdığımız lokal ropomuzu remote repomuz ilə əlaqələnirməliyik. Bunun üçün ardıcıl olaraq tək-tək aşağıdakı əmrləri yerinə yetirməliyik.



Burada, 2-ci sətirdə dırnaq işarələri ilə ayrılmış commit mətnini etdiyiniz əməliyyata uğun olaraq təyin edirsiz və 4-cü sətirdəki url sizin remote github reponuzun ünvanı olmalıdır.

Repodakı fayllarımız gitin versiya nəzarəti altına keçmədən öncə üç mərhələdən keçir:

**Untracked**: fayl mövcuddur, lakin git versiyası nəzarətinin bir hissəsi deyil.  
**Staged**: fayl git versiyasını idarə etməyə əlavə edildi, lakin dəyişikliklər təslim olunmayıb.  
**Commited**: dəyişikliklər versiya nəzarəti üçün repoya təslim olunmuşdur.



Biz fayllarımız üzərində hər hansı bir dəyişiklik və ya yeni fayl əlavə etdikdə bu dəyişikliklər bizim “working directory”də qeydə alınır və bu zaman fayllarımız git tərəfindən untracked olaraq qeyd olunur. Biz **git status**əmriniverməklə faylların hansı mərhələdə olduğuna baxa bilərik. Bundan sonra untracked fayllarımızı staging area-ya əlavə etmək üçün **git add faylın\_adı\_uzantısı\_ilə**və ya bütün faylları əlavə etmək üçün **git add .**əmrini daxil edirik. Amma yenə də fayllarımız tam olaraq gitin nəzarəti üçün təslim olunmuş sayılmır, bunun üçün biz etdiyimiz dəyişiklikləri ehtiva edən qısa mesajla **git commit -m “commit mesajı”** əmrini verməklə təslimatı tamamlamalıyq. Artıq bizim fayllarımız tam olaraq lokal repoda gitin nəzarətindədir.

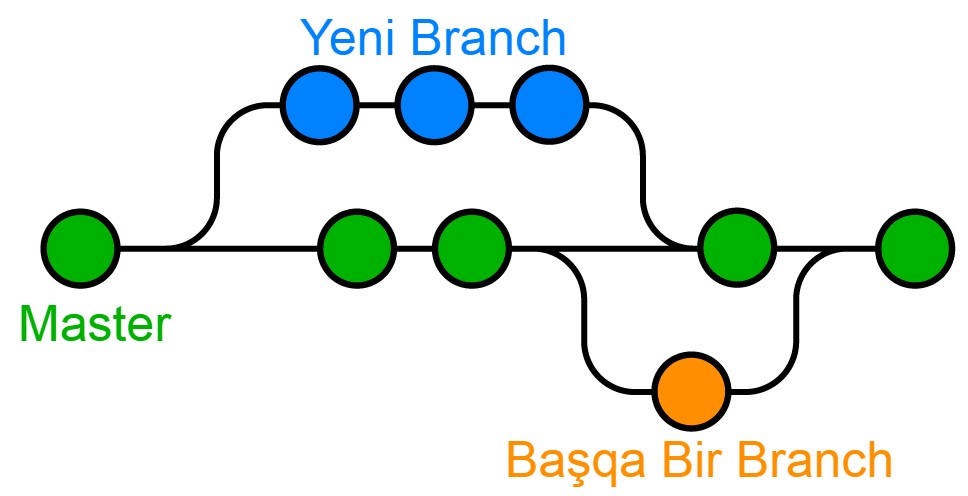
Bundan sonra biz **git push -u origin master**əmrini verməklə lokal repoda olan fayllarımızı remote repomuza yükləyə bilərik. Bu əmrdə **-u** parametrin ilk dəfə push edəndə verməklə lokaldakı branch-in müvafiq remote branch-ini təyin edirik və növbəti push proseslərində sadəcə olaraq **git push**əmri fayllarımızı müvafiq branch-a yükləməyə kifayət edəcək.

Branch anlayışı, branch-larla işləmə və git ilə bağlı daha bir çox şeyə isə [növbəti](https://medium.com/pragmatech/git-v%C9%99-github-%C9%99n-%C3%A7ox-i%CC%87stifad%C9%99-olunan-%C9%99mrl%C9%99r-part-3-49cd4f32d1de) məqaləmizdə nəzər yetirəcəyik. Vaxt ayırıb oxuduğunuz üçün təşəkkürlər.

# Git Və GitHub. Ən Çox İstifadə Olunan Əmrlər (Part 3).

[Əvvəlki](https://medium.com/pragmatech/git-v%C9%99-github-%C9%99n-%C3%A7ox-i%CC%87stifad%C9%99-olunan-%C9%99mrl%C9%99r-part-2-3cfbaa503564) məqaləmizdə gitdə ilk repomuzu yaratdıq və onu remote GitHub repomuza **push** etdik. Bu push prosesini **master** branch-i üzərindən həyata keçirdik. Gəlin indi branch anlayışının özünə baxaq və branchların bizə nə kimi imkanlar yaratdığını nəzərədn keçirdək.

**Branch**— ingilis dilindən tərcümədə budaq, şaxə, qol mənasına gəlir. Bu hərfi mənalar gitdəki branch anlayışını demək olar ki, izah edir. Proyektimizi ilk dəfə git repoya əlavə edəndə onu master barnchı üzərindən push edirik və master branchı layihənin əsas qolu olur. Biz layihədə hər hansı yenilik edəndə master branchdan yeni branchlar yaradıb dəyişikliklərimizi həmin branchlar üzərində həyata kechirdirik və hər şey qaydasında gedərsə həmin branchları yenə master brancha **merge** edirik, yəni birləşdiririk.



Repomuzda mövcud branchlara baxmaq üçün repomuz olan ünvanda terminaldan **git branch** əmrini daxil etməliyik. Bu zaman mövcud olan bütün branchların siyahısı ekranda görünəcək(hal-hazırda aktiv olan branch xüsusi işarələnmiş şəkildə). Əgər yeni branch yaratmaq istəyiriksə **git branch yeni\_branch\_adi** şəklində əmr daxil edirik. Bu zaman yeni yaranmış branch bizim cari aktiv branchı istinad olaraq götürəcək onun bir kopyasını ehtiva edən bir şaxə yaradacaq. Biz yeni yaratdığımız brancha keçid etmək üçün isə **git checkout yeni\_branch\_adi** şəklində əmr daxil edirik. Və artıq bizim working directory yeni yaratdığımız brancha dəyişəcək və etdiyimiz bütün dəyişikliklər yeni branchda həyata keçəcək. Yuxarıda əmrləri ikisini özündə birləşdirən bir əmr də vardır ki, bu əmr yeni branch yaradır və cari iş sahəsini yeni yaranmış brancha keçirir: **git checkout -b yeni\_branch\_adi**

Lokalda yaratdığımız branchı remote-a daşımaq üçün **git push -u origin yeni\_branch\_adi** şəklində əmr daxil edirik və artıq lokalda yaratdığımız branch remote olaraq da yaranmış olacaq. Yaratdığımız branch üçərində nəzərdə tutduğumuz dəyişikliklər tamamlandıqdan sonra master və yaxud da birləşdirməyi nəzərdə tutduğumuz brancha keçid etdib (checkout) **git merge yeni\_branch\_adi** əmrini daxil edirik. Bu zaman yarana biləcək kod konfliktlərini ardan qaldırmaq haqqında [buradan](https://docs.github.com/en/github/collaborating-with-issues-and-pull-requests/resolving-a-merge-conflict-using-the-command-line) baxa bilərsiniz. Merge prosesi uğurla başa çatdıqdan sonra artıq yaratdığımız lokal və remote branchları silmək üçün müvafiq olaraq aşağıdakı əmrlərdən istifadə edirik.

Git haqqında danışmalı olduğumuz digər bir məsələ **git clone** əmridir. Hər hansı bir git reposu üzərində bir neçə şəxs və ya bir şəxs bir neçə kompüter üzərindən iş görürsə, bu zaman remote repodakı faylları başqa kömpüter üzərindən də iş görməyə əlçatan etmək üçün git clone əmrindən istifadə edirik. Bunun üçün clone etmək istədiyimiz reponun url ünvanını kopyalayıb yükləmək istədiyimiz qovluqda terminaldan **git clone reponun\_urli** şəklində əmr daxil edirik. Bununla da remote ropadakı proyektin bir kopyası bizim kompüterimizdə yaranmış olur.

Biz kompüterimizdəki fayllar üzərində yenilik etməzdən əvvəl remote repoda hər hansı yeniliklər varsa onları lokal repoya əlavə etmək üçün **git pull** əmrini daxil etməliyik. Əgər biz git pull əmrini daxil etmədən lokal repoda sonradan etdiyimiz dəyişikilikləri push edə bilməyəcəyik. Buna alternativ kimi force push anlayışı (git push -f origin branch\_adi) var ki, bu zaman remote repodakı commit-lər silinəcək və lokal repodakı dəyişikliklər remote repoya yüklənəcək.

Bu və öncəki məqalələr ilə artıq git üzərində kifayət qədər əməliyyatları icra etməyi öyrənmiş olduq. Gitin digər əmr və özəllikləri barədə növbəti məqalələrimizdə danışacağıq. Vaxt ayırıb oxuduğunuz üçün təşəkkürlər.