**Xoş gəlmisiniz!**

Ssenari 1 - Faylların Öhdəliyi

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Bu ssenaridə yeni bir Git deposunu necə işə salacağınızı və faylları versiya nəzarətinə götürməyi öyrənəcəksiniz.

Versiya nəzarəti, vaxt keçdikcə bir faylda və ya sənədlər qrupunda dəyişiklikləri qeyd etməyinizi tələb edirsə, müəyyən bir versiyaya qayıtma imkanı verir. Git paylanmış bir versiya nəzarət sistemidir. Bu o deməkdir ki, ən son sənədlərin anlıq görüntüsünü almaq əvəzinə yerli maşınınızdakı anbarın tam bir güzgüsünə sahibsiniz. Anbar dəyişikliklərin nə vaxt baş verdiyini və kim tərəfindən edildiyini izləyir. Yerli maşınınızda tam bir anbarın olması şəbəkə trafikinə görə gecikmələri azaldır və əlaqəsi kəsildikdə işə davam etməyə imkan verir.

Bu Scrapbook mühiti git quraşdırılmış şəkildə əvvəlcədən qurulmuşdur. Git-in quraşdırılması ilə bağlı daha çox məlumat üçün [rəsmi sənədlərə](https://git-scm.com/book/en/v2/Getting-Started-Installing-Git) müraciət edin .

#### Addım 1 - Git Init

Bir qovluğu versiya nəzarəti altında saxlamaq üçün bir depo yaratmalısınız. Git ilə bir layihə üçün üst səviyyə qovluğunda bir depo hazırlayırsınız.

##### **Tapşırıq**

Bu yeni bir layihə olduğundan yeni bir anbarın yaradılması lazımdır. git initBir depo yaratmaq üçün əmrdən istifadə edin .

##### **ProTip**

Bir deposu işə saldıqdan sonra .git adlı yeni bir gizli alt qovluq yaradılır. Bu alt direktoriya Gitin məlumatlarını saxlamaq üçün istifadə etdiyi metadatanı ehtiva edir. Təfərrüatlarla maraqlanırsınızsa, məzmunu araşdırmaq üçün əmr satırından istifadə edin.

#### Addım 2 - Status Git

Bir qovluq anbarın bir hissəsi olduqda, bir İşləmə Kataloqu adlanır . İşləyən bir qovluq, anbardan yüklənmiş son versiyanı və hələ edilməli olmayan dəyişiklikləri ehtiva edir. Bir layihə üzərində işləyərkən bütün dəyişikliklər bu iş qovluğunda edilir.

Komandanızı istifadə edərək iş qovluğunuzla əvvəllər anbarda işləyənlər arasında hansı sənədlərin dəyişdiyini görə bilərsiniz git status.

Bu əmrin çıxışı "işləyən ağac vəziyyəti" adlanır.

#### Protip

Bütün sənədlər, əksinə deyilənə qədər Git tərəfindən "izlənilmir". Növbəti addımda necə əhatə olunduğunun təfərrüatları.

#### Addım 3 - Git əlavə edin

Sənədləri Git deposunuza saxlamaq və ya həyata keçirmək üçün əvvəlcə quruluş sahəsinə əlavə etməlisiniz . Git'in üç sahəsi, bir iş qovluğu, bir quruluş sahəsi və deponun özü var. İstifadəçilər, başqa bir şəkildə təşviq edildiyi deyilən dəyişikliklər, iş qovluğundan depoya girmədən əvvəl bir hazırlama sahəsinə keçir.

Git ilə əlaqəli əsas yanaşmalardan biri də öhdəliklərin cüzi və kiçik olmasıdır. Səhnələşdirmə sahəsi, iş kataloqunuzdakı bütün dəyişikliklər əvəzinə bir anda yalnız müəyyən faylları tanıtmağınıza imkan verərək bu iş axınının qorunmasına kömək edir.

#### Tapşırıq

Əməliyyat sahəsinə hello-world.jsgit add <file|directory> əlavə etmək üçün əmrdən istifadə edin .

Səhnələşdirmə sahəsinə bir fayl əlavə etdikdən sonra əlavə bir dəyişiklik etsəniz, fayl yenidən əlavə olunana qədər dəyişiklik əks olunmayacaqdır.

#### Protip

Adım 2-də təsvir edildiyi kimi, git statusəmr həm iş qovluğunun, həm də səhnə sahəsinin vəziyyətini zamanın istənilən nöqtəsində görməyə imkan verir.

#### Addım 4 - Git Commit

Səhnələşdirmə sahəsinə bir fayl əlavə edildikdən sonra depoya sadiq qalması lazımdır. Komanda, git commit -m "commit message"faylları quruluşdan anbara aparır və vaxt / tarix, müəllif və səhv hesabat nömrəsi kimi dəyişikliklərə əlavə kontekst və düşüncə əlavə etmək üçün istifadə edilə bilən bir mesajı qeyd edir.

Yalnız səhnələşdirmə sahəsinə əlavə edilmiş dəyişikliklər ediləcək, iş qovluğunda səhnələşdirilməmiş bütün fayllar daxil edilməyəcəkdir.

#### Tapşırıq

git commit -m "<commit message>"Mərhələli faylı işlətmək üçün istifadə edin .

#### Protip

Hər bir öhdəliyə, digər əmrlərdəki öhdəliyə qayıtmağınıza imkan verən SHA-1 qarışığı verilir.

#### Addım 5 - Git Ignore

Bəzən yerli inkişaf konfiqurasiyası kimi heç vaxt etmək istəmədiyiniz xüsusi fayllar və ya qovluqlar var. Bu faylları görməməzlikdən gəlmək üçün anbarın kökündə bir .gitignore faylı yaradırsınız .

.Gitignore fayl siz tmp uzadılması tmp ilə bütün faylları ignore edəcək \* misal üçün, ignore etmək istəyən fayllar üçün joker müəyyən etməyə imkan verir.

Müəyyən edilmiş bir joker kartla uyğun gələn hər hansı bir sənəd bir git statusçıxışda göstərilməyəcək və git addəmri sınayarkən laqeyd qalacaq .

#### Tapşırıq

Bütün \* .tmp sənədlərinə məhəl qoymamaq üçün bir .gitignore faylı əlavə edin və depoya əlavə edin .

#### Protip

.gitignoreQaydaları müxtəlif maşın arasında tətbiq təmin etmək üçün depo sadiq olmalıdır.

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Bu ssenari anbarı necə başlatdığınızı və sonra ona sənədlər verə biləcəyinizi izah etdi. Növbəti ssenaridə bu faylları necə müqayisə edəcəyimizi və dəyişikliklər edəcəyimizi araşdıracağıq. Gələcək ssenarilərdə bu dəyişiklikləri digər insanlarla necə bölüşəcəyimizi əhatə edəcəyik.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

**Xoş gəlmisiniz!**

Ssenari 2 - Dəyişikliklərin edilməsi

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Əvvəlki ssenariyə əsasən burada iş qovluğunuzdakı dəyişiklikləri necə görəcəyinizi və onları öz deposunuza necə verəcəyinizi öyrənəcəksiniz. Bu mühitdə sadiq bir sənəd olan bir Git deposu var. İşləyən qovluqda işlənmiş faylda bir dəyişiklik və icazəsiz bir sənəd mövcuddur.

Gələcək ssenarilərdə dəyişiklikləri necə geri götürə biləcəyinizi və öhdəlikləri digər insanlarla necə bölüşə biləcəyinizi izah edəcəyik.

#### Addım 1 - Status Git

Əvvəlki ssenaridə müzakirə edildiyi kimi git status, iş qovluğundakı və səhnə sahəsindəki depo ilə müqayisədə dəyişiklikləri görməyə imkan verir.

Cari depoda işləyən məlumatları nəzərə alsaq, git statusiş kataloqumuzda əvvəlcədən işlənmiş bir fail.js sənədində bir dəyişiklik edildiyini, lakin hələ səhnələşdirmə sahəsinə köçürülmədiyini göstərir.

#### Addım 2 - Fərqli Git

Komanda git diffiş kataloqundakı dəyişiklikləri əvvəlcədən hazırlanmış bir versiya ilə müqayisə etməyə imkan verir. Varsayılan olaraq, əmr işləyən qovluğu və HEAD öhdəliyini müqayisə edir .

Daha köhnə bir versiya ilə müqayisə etmək istəsəniz, məsələn, parametri olaraq törətmə qarışığını təmin edin git diff <commit>. Öhdəliklərlə müqayisə, dəyişdirilmiş bütün fayllar üçün dəyişiklikləri çıxaracaqdır. Dəyişiklikləri tək bir faylla müqayisə etmək istəyirsinizsə, bu kimi bir arqument kimi ad verin git diff committed.js.

#### Protip

Varsayılan olaraq çıxış birləşmiş diff formatındadır. Komanda git difftool, fərqləri görmək üçün seçdiyiniz xarici bir aləti yükləyəcəkdir.

#### Addım 3 - Git əlavə edin

Dəyişiklik etmək üçün əvvəlki ssenaridə olduğu kimi əvvəlcə git addəmrdən istifadə olunaraq səhnələşdirilməlidir .

#### Tapşırıq

Dəyişikliklərinizi istifadə edərək birinci mərhələyə keçin git add

#### Protip

Faylların adını dəyişdirirsinizsə və ya silsəniz, izləmək üçün bu əmrləri əlavə etmək əmrində göstərməlisiniz. Əməliyyatı yerinə yetirmək və səhnə sahəsini yeniləmək üçün istifadə edə biləcəyiniz git mvvə git rmgit üçün alternativlər .

#### Addım 4 - Mərhələli Fərqlər

Dəyişikliklər quruluş zonasında olduqdan sonra çıxışı göstərməyəcəkdir git diff. Varsayılan olaraq, git diffyalnız iş qovluğunu müqayisə edəcəkdir, səhnə sahəsini deyil.

Səhnələşdirmə sahəsindəki dəyişiklikləri əvvəlki öhdəliklə müqayisə etmək üçün mərhələli parametr təqdim edirsiniz git diff --staged. Bu, bütün dəyişikliklərinizi düzgün qurduğunuzdan əmin olmağa imkan verir.

#### Addım 5 - Git Log

Bu əmr git loganbarın tarixçəsinə və öhdəlik jurnalına baxmağa imkan verir.

#### Protip

Günlük çıxışı formatı çox çevikdir. Məsələn, hər bir öhdəliyi bir sətirdə çıxarmaq üçün əmr verilir git log --pretty=format:"%h %an %ar - %s". Daha çox təfərrüatları istifadə edərək giriş log man səhifəsində tapa bilərsinizgit log --help

#### Addım 6 - Git Show

Git log sizə öhdəlik müəllifini və mesajını bildirirsə də, öhdəlikdəki dəyişiklikləri görmək üçün əmrdən istifadə etməlisiniz git show

Digər əmrlərdə olduğu kimi, varsayılan olaraq HEAD əmrindəki dəyişiklikləri göstərəcəkdir. git show <commit-hash>Köhnə dəyişiklikləri görmək üçün istifadə edin .

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Bu məşq dəyişikliklərinizə necə baxa biləcəyinizi və anbara necə verə biləcəyinizi nümayiş etdirdi. Növbəti ssenaridə bu dəyişiklikləri digər insanlarla necə paylaşacağımızı araşdıracağıq.

Bu məşq istədiyiniz nöqtədə yazdığınız nümunəni və kodu nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

**Xoş gəlmisiniz!**

Ssenari 3 - Uzaqdan işləmək

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Bu ssenaridə anbarınızdakı dəyişiklikləri digər insanlarla necə bölüşdürə biləcəyinizi və dəyişikliklərini öz anbarınıza birləşdirə biləcəyinizi öyrənəcəksiniz. Bu mühit, tək bir öhdəliyi olan bir Git deposu ilə konfiqurasiya edilmişdir. Uzaqdan bir depo konfiqurasiya edildi, lakin yerli depo ilə əlaqələndirilmədi.

Gitin paylanmış bir versiya idarəetmə sistemi olması ilə, yerli depoda anbarın başlanğıcından bəri edilən bütün qeydlər, fayllar və dəyişiklikləri ehtiva edir. Hər kəsin ən son versiya üzərində işləməsini təmin etmək üçün dəyişikliklərin paylaşılması lazımdır. Bu dəyişiklikləri digər depolarla bölüşərkən yalnız fərqlər prosesi son dərəcə sürətli edərək sinxronlaşdırılacaqdır.

#### Addım 1 - Git Remote

Uzaqdan depolar, depodan və ya dəyişikliyi paylaşma imkanı verir. Uzaq yerlər ümumiyyətlə bir qurma sunucusu, bir komanda üzvləri maşını və ya Github.com kimi mərkəzləşdirilmiş bir mağazadır. Uzaqdan idarəolunangit remote ad və uzaq bir yer olan komanda, tipik olaraq bir HTTPS URL və ya bir SSH bağlantısı, məsələn [*https://github.com/OcelotUproar/ocelite.git*](https://github.com/OcelotUproar/ocelite.git) və ya git@github.com : / OcelotUproar / ocelite istifadə edərək əlavə olunur. git .

Səmimi ad digər əmrlərdəki yerə müraciət etməyə imkan verir. Yerli deponuz, ssenarinizdən asılı olaraq bir çox fərqli uzaq anbarlara istinad edə bilər.

#### Tapşırıq

Bu mühitin / s / remote-project / 1 uzaq bir depo yeri var . Istifadə edərək git remote, adın mənşəyi ilə bu uzaq yeri əlavə edin .

#### Protip

git cloneGələcək bir ssenaridə müzakirə olunan istifadə edirsinizsə, klonlaşdırdığınız yer avtomatik olaraq ad mənşəli uzaqdan idarəedici olaraq əlavə ediləcəkdir .

#### Addım 2 - Git Push

Siz öhdəliyini bölüşmək üçün hazır olduğunuzda sizə lazım təkan vasitəsilə uzaq depo onları git push. Tipik bir Git iş axını, bir tapşırığı yerinə yetirərkən bir çox kiçik öhdəliklər yerinə yetirmək və komanda daxilində kodun sinxronizasiyasını təmin etmək üçün müvafiq nöqtələrdə məsafəni itələmək, məsələn tapşırıq tamamlandıqda.

git pushKomanda iki parametrləri izləyir. İlk parametr, ilk addımda müəyyənləşdirdiyimiz uzaq deponun dost adıdır. İkinci parametr filialın adıdır. Varsayılan olaraq, bütün git depolarında kodun işlədildiyi bir ana filial var.

#### Tapşırıq

Əsas filialdakı öhdəlikləri mənşə pultuna itələyin.

#### Addım 3 - Git Pull

Harada git pushbir uzaq deposu dəyişikliklər təkan verir, git pulləks moda işləyir. git pulluzaqdakı bir depodan yerli versiya ilə dəyişiklikləri sinxronlaşdırmağa imkan verir.

Uzaq depodakı dəyişikliklər avtomatik olaraq hazırda işlədiyiniz filiala birləşir.

#### Tapşırıq

Dəyişiklikləri pultdan master filialınıza çəkin.

Növbəti addımda hansı dəyişikliklərin edildiyini araşdıracağıq.

#### Addım 4 - Git Log

Əvvəlki ssenaridə təsvir olunduğu kimi git log, anbarın tarixçəsini görmək üçün əmrdən istifadə edə bilərsiniz . git showKomanda hər bir törətmək dəyişikliklər keçirmək üçün imkan verir.

Bu nümunədə, git log" DifferentUser@JoinScrapbook.com " tərəfindən " Hata # 1234 üçün Düzelt" mesajı ilə yeni bir öhdəlik göstərilir . Çıxışda git showyaşıl rəngdə fayla əlavə edilmiş yeni sətirlər vurğulanır.

#### Protip

git log --grep="#1234"1234 saylı bütün öhdəlikləri tapmaq üçün əmrdən istifadə edin

**Addım 5 - Get Get**

Komanda git pulliki fərqli əmrin birləşməsidir git fetchvə git merge. Fetch, dəyişiklikləri uzaqdakı depodan *uzaqdan / <remote-name> / <remote-branch-name>* adlı ayrı bir filiala *yükləyir* . Şöbə istifadə edərək əldə edilə bilər git checkout.

İstifadə git fetch, mövcud filialınıza təsir etmədən dəyişiklikləri nəzərdən keçirmək üçün əla bir yoldur. Filialların adlandırma formatı kifayət qədər çevikdir ki, eyni adlı birdən çox məsafədən və budaqdan istifadə edə və aralarında asanlıqla keçə bilərsiniz.

Aşağıdakı əmr gətirilən dəyişiklikləri masterə birləşdirəcəkdir.

git merge remotes/<remote-name>/<remote-branch-name> master

Gələcək bir ssenaridə birləşməni daha ətraflı izah edəcəyik.

**Tapşırıq**

*Mənşə* deposunda əlavə dəyişikliklər edilmişdir . git fetchDəyişiklikləri yükləmək üçün istifadə edin və sonra onları görmək üçün filialı çıxartın.

**Protip**

Əmrdən istifadə edərək bütün uzaq filialların siyahısına baxa bilərsiniz git branch -r

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Bu ssenari, fərqli anbarlar arasında dəyişiklikləri necə itələyə / çəkə biləcəyinizi nümayiş etdirdi. Davamlı istifadə edərək git pushvə git pullhər kəsin kod bazasının ən son versiyasını əldə etməsini təmin edə bilərsiniz.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

**Xoş gəlmisiniz!**

Ssenari 4 - Dəyişikliklərin Geri Alınması

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Bir versiya idarəetmə sistemindən istifadə edərkən əsas üstünlüklərdən biri dəyişiklikləri geri qaytarmaq və əvvəlki versiyaya qayıtmaqdır. Git, güclü bir yanaşma təmin edir və deponuzu idarə etməyə nəzarət edir və bu ssenaridə araşdıracağımız tarixdir.

Bu ssenaridə mühit bir sıra sadə öhdəliklərin olduğu bir Git deposu ilə yaradılmışdır.

#### Addım 1 - Git Checkout

Git ilə işləyərkən ümumi bir ssenari iş qovluğundakı dəyişiklikləri geri qaytarmaqdır. Komanda git checkoutiş qovluğundakı hər şeyi son işlənmiş versiyaya əvəz edəcəkdir.

Bütün faylları dəyişdirmək istəyirsinizsə , cari qovluğu göstərmək üçün bir nöqtə ( . ) İstifadə edin , əks halda boşluqlarla ayrılmış qovluqları / faylları siyahıya alın.

#### Tapşırıq

İşləmə git checkoutqovluğundakı dəyişiklikləri silmək üçün istifadə edin .

#### Addım 2 - Git Sıfırla

Bir öhdəliyin ortasındasınızsa və səhnələşdirmə sahəsinə fayllar əlavə etdiniz, lakin sonra fikrinizi dəyişdirdinizsə, git resetəmrdən istifadə etməlisiniz. git resetfaylları quruluş sahəsindən iş qovluğuna geri köçürəcəkdir. Bütün faylları sıfırlamaq istəyirsinizsə, a istifadə edin . cari qovluğu göstərmək üçün, əks halda boşluqlarla ayrılmış faylları siyahıya alın.

Bu, öhdəliklərinizi kiçik və fokuslu saxlamağa çalışarkən çox faydalıdır, çünki çox sayda əlavə etmisinizsə faylları yenidən səhnə sahəsindən çıxara bilərsiniz.

#### Tapşırıq

Dəyişiklikləri səhnələşdirmədən istifadə edərək yenidən iş qovluğuna köçürün git reset

#### Addım 3 - Sıfırlayın

A git reset --hardbir komanda git yenidən və git checkout həm birləşdirmək olacaq. Nəticə, səhnələşdirmə sahəsindən çıxarılan fayllar olacaq və iş qovluğu son işləmə vəziyyətinə qaytarılır.

#### Tapşırıq

Dəyişiklikləri həm quruluş sahəsindən, həm də iş qovluğundan istifadə edərək silin git reset

#### Protip

İstifadə HEADvəziyyəti son öhdəliyə qaytaracaq, istifadə git reset --hard <commit-hash>etmək hər hansı bir öhdəlik vəziyyətinə qayıtmağa imkan verir. Unutmayın, HEAD, filialın son əməli-hash üçün bir təxəllüsdür.

#### Addım 4 - Git Revert

Əgər siz əvvəllər sənədlər git reverthazırlamısınız, ancaq səhv etdiyinizi başa düşmüsünüzsə, əmr sizə verilənləri ləğv etməyə imkan verir. Komanda, qaytarılmış öhdəliyin tərs təsirini göstərən yeni bir öhdəlik yaradır.

Dəyişikliklərinizi itələməmisinizsə git reset HEAD~1, eyni təsirə sahibsiniz və son öhdəliyi aradan qaldıracaqsınız.

#### Tapşırıq

git revertSon öhdəlikdəki dəyişiklikləri geri qaytarmaq üçün istifadə edin .

Qeyd edək ki, hər bir öhdəlik üçün öhdəlik mesajı yaratmaq üçün Vim redaktoru sessiyası açılacaqdır. Öhdəlik mesajını saxlamaq və vimdən çıxmaq üçün hər bir Vim seansı üçün əmr yazın : wq .

#### Protip

Yeni öhdəliklər yaratmağın motivasiyası Git-də tarixin yenidən yazılmasının anti-nümunə olmasıdır. Əgər öhdəliklərinizi itələmisinizsə, dəyişiklikləri ləğv etmək üçün yeni öhdəliklər yaratmalısınız, çünki digər insanlar bu müddətdə öhdəlik götürmüş ola bilər.

#### Addım 5 - Git Revert

Birdən çox öhdəliyi bir anda geri qaytarmaq üçün ~ işarəsini mənfi mənasına gətiririk. Məsələn, HEAD ~ 2 başdan iki əmrdir. Bu simvollarla birləşdirilə bilər ... iki iş arasında demək.

#### Tapşırıq

git revert HEAD...HEAD~2HEAD və HEAD ~ 2 arasındakı öhdəlikləri geri qaytarmaq üçün əmrdən istifadə edin .

#### Protip

git log --onelineÖhdəlik tarixçəsinə qısa bir baxış üçün əmrdən istifadə edə bilərsiniz .

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Bu ssenari, etdiyiniz dəyişiklikləri necə sıfırlaya və geri qaytara biləcəyinizi və əvvəlki vəziyyətə necə qayıda biləcəyinizi nümayiş etdirdi.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

**Xoş gəlmisiniz!**

Ssenari 5 - Çatışmaların Birləşdirilməsi

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Bu ssenaridə anbarınızdakı dəyişiklikləri digər insanlarla necə bölüşə biləcəyinizi və dəyişiklikləri öz anbarınızda necə birləşdirə biləcəyinizi öyrənəcəksiniz.

Bu ssenarinin tamamlanmasına kömək etmək üçün uzaq bir depo əlavə edilmiş bir depo yaradılmışdır. Ətraf mühit, başqa bir geliştiricinin sizinlə eyni fayl üzərində işlədiyini simulyasiya etdi. Nəticədə, onların dəyişikliklərini çəkdiyiniz zaman hər iki dəyişiklik arasında ziddiyyət olduğunu görürsünüz. Bu ssenari bu münaqişənin necə həll ediləcəyini izah edir.

#### Addım 1 - Birləşdirin Git

git fetchYoxlanılır və birləşməsi edilə bilər bir ayrı-ayrı filialı komanda downloads dəyişikliklər. Birləşmə zamanı Git, borcları avtomatik olaraq birləşdirməyə çalışacaq.

Heç bir ziddiyyət olmadıqda birləşmə 'sürətli yönləndiriləcək' və heç bir şey etmək məcburiyyətində qalmayacaqsınız. Bir ziddiyyət varsa, bir səhv alacaqsınız və anbar birləşmə vəziyyətində olacaq.

#### Tapşırıq

Ətrafınızdakı uzaq bir depodan dəyişikliklər gətirildi.

İndi mənşə / master dəyişikliklərini birləşdirməlisiniz.

Bu birləşmə münaqişəsi ilə nəticələnəcəkdir. Münaqişə, birləşdirilmənin uğursuz olduğunu göstərir, çünki hər iki depo da faylı əlavə edib. Bunu növbəti addımlardan sonra həll edəcəyik.

#### Protip

Öhdəlikləri kiçik və diqqət mərkəzində saxlayaraq birləşmə münaqişəsi ehtimalını azaldırsınız.

Komanda və git pull-nin birləşməsidir .fetchmerge

#### Addım 2 - Münaqişəyə baxmaq

Ziddiyyət meydana gəldikdə həm yerli, həm də uzaqdan dəyişikliklər eyni faylda unix diff formatında görünəcəkdir. Bu, git diff tərəfindən istifadə edilən eyni formatdır .

Format oxumaq üçün yerli dəyişikliklər arasında üst görünür <<<<<<< RƏHBƏRİ və ======= uzaq dəyişikliklər arasında altında olan ======= və >>>>> >> məsafələr / mənşə / usta .

Münaqişəni həll etmək üçün sənədlər istədiyimiz son vəziyyətə uyğun şəkildə düzəldilməlidir. Növbəti addımda bunu nümayiş etdirəcəyik.

#### Protip

Git, münaqişələrin həllini asanlaşdırmaq üçün fərqli komanda xəttini və vizual birləşmə vasitələrini dəstəkləyir. Komanda git mergetoolxarici bir vasitə işə salacaq, biz [kdiff3-ün](http://kdiff3.sourceforge.net/) böyük pərəstişkarlarıyıq .

#### Addım 3 - Münaqişənin həlli

Münaqişə düzeltmek üçün sadə yolu istifadə və ya yerli və ya uzaq versiyası almaq üçün git checkout --ours staging.txtvə ya git checkout --theirs staging.txt. Daha çox nəzarətə ehtiyacınız varsa, faylları normal qaydada düzəldə bilərsiniz.

Əllər və ya git checkout istifadə edərək, sənədlər istədiyiniz vəziyyətdə olduqda, mərhələyə ehtiyacınız var və dəyişiklikləri həyata keçirin. Varsayılan öhdəlik verildikdə birləşmə təfərrüatları ilə və hansı faylların ziddiyyətli olduğu mesajı yaradılacaqdır.

#### Tapşırıq

Uzaqdakı dəyişiklikləri seçərək ziddiyyəti həll edin və git addardından birləşdirməni tamamlayın git commit.

#### Protip

Birləşmənin ortasında qayıtmaq və yenidən cəhd git reset --hard HEAD;etmək istəyirsinizsə, əvvəlki vəziyyətinizə qayıtmaq üçün əmrdən istifadə edin .

git commit --no-editVarsayılan öhdəlik mesajını istifadə etmək istədiyiniz zaman istifadə edin .

#### Addım 4 - Sürətli olmayan irəli

Sürətli olmayan irəli birləşdirməni simulyasiya etmək üçün aşağıdakılar baş verdi.

1) Developer A Developer B-dən ən son dəyişiklikləri çıxarır.  
2) B B yerli deposunda dəyişikliklər edir.  
3) A Developer, yerli deposunda ziddiyyətli olmayan dəyişikliklər edir.  
4) A Developer, Developer B-dən son dəyişiklikləri çıxarır.

Bu ssenaridə Git, Geliştirici B-dən bir neçə dəyişiklik etdiyinə görə dəyişiklikləri sürətli ötürə bilmir.

Bu baş verdikdə, Git dəyişiklikləri avtomatik birləşdirməyə çalışacaq. Əgər heç bir ziddiyyət yoxdursa, birləşmə başa çatacaq və birləşmənin həmin vaxt baş verdiyini göstərmək üçün yeni bir öhdəlik yaradılacaqdır.

Birləşmələr üçün standart öhdəlik mesajı "Filialı birləşdir '''of ". Bu vəzifələr anbarlar arasındakı sinxronizasiya nöqtələrini göstərmək üçün faydalı ola bilər, həm də səs-küylü bir qeyd jurnalını yaradır. Növbəti addımda alternativ yanaşmaları araşdıracağıq.

#### Tapşırıq

Dəyişiklikləri uzaqdakı depodan çəkin və aşağıdakı əmrdən istifadə edərək standart borc mesajını istifadə edin.

git pull --no-edit origin master

Öhdəliklərə baxa bilərsiniz git log --all --decorate --oneline

#### Adım 5 - Git Rebase

Birləşdirmə mesajları sinxronizasiya nöqtələrini göstərmək üçün faydalı ola bilər, lakin eyni zamanda çox səs çıxara bilər. Məsələn, yerli filiallara qarşı işləyirsinizsə və itələməmisinizsə, bu əlavə məlumat anbara baxan digər inkişaf etdiricilər üçün mənasız və qarışıqdır.

Bunu həll etmək üçün git rebaseəvəzinə istifadə edə bilərsiniz git merge. Yenidənqurma, etdiyiniz dəyişiklikləri açacaq və dəyişiklikləri eyni filialda olduğu kimi tətbiq edərək filialdakı dəyişiklikləri yenidən oynayacaq. Nəticə birləşmə üçün təmiz bir tarix və qrafikdir.

Mühüm kimi rebasehər bir yeni hash id olacaq törətmək yerinə birləşməsi dəyişikliklər replay olacaq. Siz və ya digər inkişaf etdiricilər anbarı itələyib / çəkmisinizsə, tarix dəyişdirmək öhdəlikləri itirmək üçün gedə bilər. Beləliklə, ictimaiyyətə açıqlanan öhdəlikləri geri qaytarmamalısınız, məsələn öhdəlikləri itələmək, sonra fərqli bir filialdan daha əvvəl götürdüyünüz vəzifələri geri götürmək. Nəticə əvvəllər fərqli hash kimliklərinə sahib olan ictimai vəzifələr olacaqdır. Daha çox təfərrüatı, [The Reils of Rebasing-](https://git-scm.com/book/ch3-6.html#The-Perils-of-Rebasing) də tapa bilərsiniz .

#### Addım 6 - Çəkmə İstəklərini Azaltma

Bu yanaşma uzaq filiallarla işləyərkən də tətbiq olunur və aşağıdakılardan istifadə edərək bir çəkmə sorğusu verilərkən tətbiq oluna bilər.

git pull --rebase

Bu, hər bir öhdəliyinizdən əvvəl bir çəkmə istəyi etmiş kimi davranacaqsınız.

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Bu ssenaridə birləşmələrin idarə edilməsinin müxtəlif yollarını araşdırdıq. Biz necə istifadə gördüm git fetchvə git mergebizim git log saxlamaq və istifadə təmiz öhdəliyini necə nəhayət digər öhdəliyini və münaqişələrin həlli necə uzaq dəyişikliklər çəkmək üçün git rebase. Birləşmə Git-in vacib bir hissəsidir, gələcək ssenarilərdə daha dərindən araşdıracağımız bir mövzu.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

**Xoş gəlmisiniz!**

Ssenari 6 - Budaqlardan istifadə olunan təcrübələr

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Bu ssenaridə anbarınızda budaqları necə yarada biləcəyinizi öyrənəcəksiniz. Filial, yeni bir iş qovluğunda effektiv şəkildə işləməyə imkan verir. Nəticə budur ki, tək bir Git deposu, hər biri qovluqları dəyişdirmədən dəyişdirilə bilən bir çox fərqli kod bazasına sahib ola bilər.

Git-dəki standart filiala *master* deyilir . Əlavə filiallar, dəyişikliklər etmək, birləşdirmək və itələmək kimi usta olduğu kimi eyni əməliyyatları və əmrləri yerinə yetirməyə imkan verir. Əlavə filiallar usta ilə eyni şəkildə işlədikləri üçün prototip və təcrübələr üçün idealdırlar, çünki hər hansı bir dəyişiklik tələb olunarsa master üçün birləşdirilə bilər.

Bir filialı dəyişdirdiyiniz zaman, Git iş qovluğunun tərkibini dəyişdirir. Bu, fərqli filialları və ya yerləri əks etdirmək üçün hər hansı bir konfiqurasiyanı və ya parametrləri dəyişdirməyə ehtiyacınız olmadığı anlamına gəlir.

Bu mühit uzaq *mənşəli bir* depo müəyyən edilmiş bir depo ilə konfiqurasiya edilmişdir .

#### Addım 1 - Git Şöbəsi

Filiallar ümumiyyətlə master olan başqa bir filiala əsaslanaraq yaradılır. Komanda git branch <new branch name> <starting branch>mövcud bir filialı alır və işləmək üçün ayrı bir filial yaradır. Bu nöqtədə hər iki filial eynidir.

Budağa keçmək üçün git checkout <new branch name>əmrdən istifadə edirsiniz .

#### Tapşırıq

'New\_branch' adlı yeni bir filial yaradın və ödəmə edin

#### Protip

Komanda git checkout -b <new branch name>yeni yaradılan filialı yaradır və ödəməlidir.

#### Addım 2 - Filialların siyahısı

Bütün filialların siyahısını vermək üçün əmrdən istifadə edin git branch.

Əlavə arqument -auzaq filialları, filial -vüçün HEAD törətmə mesajını da daxil edər

#### Tapşırıq

Bütün filialları son istifadə mesajı istifadə edərək sadalayın git branch -va

#### Addım 3 - Ustaya Birləşdirin

Yeni filiala öhdəlik verilib. Bu birləşməsi master ilk bu halda ustası, hədəf filialı çıxış, və sonra bir filialının öhdəliyini ilə birləşməsi üçün 'git birləşməsi "funksiyanı istifadə etmək lazımdır.

#### Tapşırıq

Yeni filialınızdakı öhdəlikləri yenidən masterə birləşdirin.

#### Addım 4 - Dalları itələyin

Əvvəlki kurslarda müzakirə etdiyimiz kimi, budağı bir məsafədən uzaqlaşdırmaq istəyirsinizsə əmri istifadə edin git push <remote\_name> <branch\_name>

#### Addım 5 - Filialları təmizləyin

Budaqların təmizlənməsi səs-küy və qarışıqlığı aradan qaldırmaq üçün vacibdir. Bir filialı silmək -düçün, məsələn , arqument verməlisinizgit branch -d <branch\_name>

#### Tapşırıq

İndi filial masterə birləşdirilib, artıq tələb olunmur. Anbarınızı təmiz və başa düşmək üçün yeni filialınızı silin.

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Bu ssenaridə, prototip və təcrübələr üçün ideal olan budaqlarla necə işləyə biləcəyinizi araşdırdıq, çünki onlar tez yaradıla və atıla bilər.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

**Xoş gəlmisiniz!**

Ssenari 7 - Hata tapmaq

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Proqram mövcud olduğu müddətdə proqram səhvləri problemdir. Git anbardakı bütün öhdəlikləri qeyd etdikdə, məsələlərin necə təqdim olunduğunu müəyyənləşdirərkən böyük bir məlumat mənbəyi və diaqnostik vasitə olur.

Bu ssenaridə problemi ortaya qoyan müxtəlif yolları araşdıracağıq. Ətraf mühit, elementlərin siyahısını əks etdirən tək bir HTML faylı olan bir Git deposu ilə işə salındı.

#### Git Diff Two Commiss

git diffKomanda öhdəliyini arasında dəyişib nə müqayisə üçün sadə edir. İki öhdəlik arasındakı fərqləri ortaya çıxaracaqdır.

#### Misal

İki əmələ gələn hash-id və ya göstəriciləri (blob) təqdim edərək hər iki işi görməli şəkildə edə bilərsiniz.

git diff HEAD~2 HEAD

#### Git Log

git logÖhdəlik mesajlarını görməyinizə kömək etsə də , default olaraq həqiqətən dəyişənləri çıxartmır. Şükürlər olsun ki, əmr son dərəcə çevikdir və əlavə seçimlər anbarın tarixinə dair faydalı məlumatlar verir.

#### Nümunələr

Qısa bir görünüşdə öhdəliklərin ümumi görünüşünü görmək üçün əmrdən istifadə edin git log --oneline

Öhdəlik məlumatlarını dəyişən şeyin fərqləri ilə çıxarmaq üçün -p istemini daxil etməlisinizgit log -p

Bu tarixin hamısını çıxardacaq. Bir sıra fərqli seçimlərlə süzə bilərsiniz. N <sayı> baş ekran öhdəliyini bir limit müəyyən edir. Məsələn git log -p -n 2HEAD və HEAD ~ 1 göstərilir.

Zaman müddətini bilirsinizsə, --since = "2 həftə əvvəl" və \_-- qədər = "1 gün əvvəl" istifadə edərək müəyyən bir tarixdən əvvəl öhdəliklər arasında bir müddət istifadə edə bilərsiniz .

Əksər Git əmrlərində HEAD...HEAD~1olduğu kimi, terminalda göstərildiyi kimi bir sıra öhdəliklər çıxara bilərik .

Komanda istifadə git log --grep="Initial"mesajında ​​"Başlanğıc" sözünü daxil edən bütün öhdəlikləri çıxaracaqdır. Öhdəlikləri səhv izləmə nömrələri ilə etiketləsəniz, bu faydalıdır.

#### Protip

Birləşmə ssenarisində müzakirə etdiyimiz kimi, birləşmə bildiriş öhdəliklərindən istifadə etdiyiniz üçün öhdəlik tarixçəniz səs-küylü ola bilər. Bunları silmək üçün -m arqumentini git log ilə təmin edin .

#### Git Bisect

git bisectƏmrləri problemi və reqressiya təqdim törətmək olan depo bir ikili axtarış axtarır etmək üçün imkan verir. Bu addımda list.html-də HTML etiketlərini unutduran öhdəliyi tapacağıq.

Git bisect bir sıra addımlar atır, nəticələri görmək üçün addımları atın.

#### Addımlar

1. Bisekt rejiminə daxil olmaq üçün əmrdən istifadə edirsiniz git bisect start.
2. Bir dəfə iki hissəli rejimdə cari ödəmənizi pis istifadə kimi təyin edirsiniz git bisect bad. Bu, nə vaxt tətbiq edildiyini görmək üçün axtardığınız problemi ehtiva etdiyini göstərir.
3. Pis bir öhdəliyin harada baş verdiyini müəyyən etdik, indi son bilinən yaxşı öhdəliyin nə vaxt istifadə olunduğunu təyin etməliyik git bisect good HEAD~5. Bu vəziyyətdə beş iş əvvəl idi.
4. Addım 3 pis və yaxşı öhdəliklər arasındakı öhdəliyi yoxlayacaqdır. Daha sonra öhdəliyi yoxlaya, səhvlərin olub olmadığını yoxlamaq üçün testlər apara və s. Bu nümunədə istifadə edərək məzmunu yoxlaya bilərsinizcat list.html
5. Hər şeyin düzgün HTML etiketi olduğu üçün bu öhdəlik yaxşı görünür. Git'e istifadə etdiyimiz üçün xoşbəxt olduğumuzu söyləyirik git bisect good. Bu, addım 5-də və pis öhdəliyimizdə göstərildiyi kimi son bilinən yaxşı öhdəliyin ortasında avtomatik olaraq öhdəliyi yoxlayacaqdır.
6. Əvvəllər etdiyimiz kimi öhdəliyin yaxşı və ya pis olduğunu yoxlamalıyıq. cat list.html
7. Bu öhdəliyin itkin HTML etiketi var. İstifadəsi git bisect badaxtarışın sona çatması və əlaqəli törədici kimliyini çıxartmasıdır.

Nəticə budur ki, beş vəzifəni araşdırmaq əvəzinə, yalnız ikisini axtardıq. Çox daha böyük bir timescale On Böleceksiniz sizə signifant zaman saxlaya bilərsiniz.

#### Git günahlandırır

"Günahlandırma" mədəniyyətinə sahib olmaq arzuolunmaz olsa da, gələcəkdə inkişaflara kömək etmək üçün sənədin müəyyən hissələrində kimin işlədiyini bilmək faydalı ola bilər. Bu harada git blamekömək edə bilər.

git blame <file> bir sənədin hər sətrini son düzəliş edən revizyonu və müəllifi göstərir.

#### Misal

Bir faylı günahlandırmaq hər sətirə son toxunan şəxsin çıxışı olacaq.

git blame list.html

Narahat olduğumuz sətirləri bilsək, çıxmaq üçün bir sıra sətir təmin etmək üçün -L parametrindən istifadə edə bilərik .

git blame -L 6,8 list.html

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Git deposu içindəki mənbə kodunuzun hərtərəfli tarixçəsi ilə keçmişə qayıtmağa və böcəklər və ya performans məsələləri kimi problemlərin baş verdiyi vaxtı müəyyənləşdirməyə imkan verir. Kimi əmrlər git diffdəyişiklikləri tez bir zamanda müqayisə etməyə imkan verir git bisect, səbəbi müəyyənləşdirmək üçün axtarışa kömək edir.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

# Xoş gəlmisiniz!

#### Ssenari 8 - Git ilə Picking

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Kiçik öhdəliklərin üstünlüklərindən biri də hansını birləşdirmək istədiyinizi seçməyinizdir.

Bu problem, xüsusən də uzun ömürlü, əsas filialla köhnəlmiş filiallara aiddir, nəticədə sadəcə birləşmək üçün çox sayda ziddiyyət yaranır. Bu, çox aktiv açıq mənbəli layihələrdə çox rast gəlinən bir hadisədir.

Bu baş verdikdə fərdi öhdəlikləri seçib sadəcə əsas filiala birləşdirmək istəyərsiniz.

Ətraf mühit ayrı bir filialda hazırlanmış bir HTML siyahısı olan bir depo ilə konfiqurasiya edilmişdir. Bu ssenaridə master olmaq istədiyimiz bu filialdan müəyyən öhdəliklər seçəcəyik.

**Addım 1 - Albalı yığımı**

Bu ssenaridə, iki html faylı yaratmış \_new *filialında* bir sıra öhdəliklər var. Bu ssenaridə yalnız sənədlərdən birindəki dəyişikliklər bizi maraqlandırır, lakin filialı birləşdirsək, beş öhdəliyi və istənməyən dəyişiklikləri birləşdirərik.

Fərdi öhdəlikləri birləşdirmək üçün git cherry-pick <hash-id|ref>əmrdən istifadə edirik . Bu , *birləşmək* üçün oxşar bir şəkildə davranır , əgər heç bir ziddiyyət yoxdursa, öhdəlik avtomatik olaraq birləşdiriləcəkdir.

**Tapşırıq**

Maraqlandığımız üç öhdəliyi birləşdirmək üçün aşağıdakı addımları izləyin. Öhdəlikləri orijinal depoda baş verdikləri qaydada yenidən oxumaq istəyirik.

1. "İlkin öhdəlik, maddə yoxdur" mesajı ilə öhdəliyi Cherrypick
2. Cherrypick "İlkin siyahı" mesajı ilə öhdəlik götürün
3. Cherrypick "siyahıya son maddələr əlavə olunur" mesajı ilə öhdəlik götürün

**Protip**

Mövcud filialın ucunu göstərmək üçün əvvəllər HEAD istifadə etdik. Sintaksisdən istifadə edərək filiallara müraciət edə bilərsiniz \_~ # \_. Məsələn new\_branch~3, filialdakı ikinci-sonuncu işə aiddir, bu halda "İlkin öhdəlik, maddə yoxdur" öhdəlik mesajı var

#### Addım 2 - Albalı toplama münaqişəsinin həlli

Eyni şəkildə birləşmə qarşıdurmalarla nəticələnə bilər, albalı yığımı da ola bilər. Siz özünüz ya filialı birləşmə faylları təyinat və ya seçilməsi ilə eyni şəkildə münaqişələrin həlli onların və ya bizim vasitəsilə git checkout.

Bir səhv etdiyiniz kimi hiss edirsinizsə, seçiminizi dayandıraraq geri qaytara bilərsiniz git cherry-pick --abort

#### Tapşırıq

Albalı istifadə edərək ikinci siyahının yaradılmasını istifadə edin git cherry-pick new\_branch~1

Bu birləşmə münaqişəsi ilə nəticələnəcəkdir. Münaqişəni istifadə edərək həll edin git checkoutvə seçilmiş öhdəliyi seçin.

Albalı seçimini tamamladığımız yerə hazır olduqda davam edin.

#### Addım 3 - Qarşıdurmadan sonra albalı yığmağa davam edin

Münaqişələr həll edildikdən sonra əmrdən istifadə edərək albalı seçiminə davam edə bilərsiniz git cherry-pick --continue.

Birləşdirmədən istifadə etmək kimi , albalı seçimini həll etmək öhdəliklə nəticələnəcəkdir.

#### Tapşırıq

Əvvəllər ziddiyyətli maddəni əlavə edərək, - davam et seçimini istifadə edərək albalı seçimini tamamlayın .

git add list2.html

git cherry-pick --continue

Bu zaman standart redaktor, bu vəziyyətdə vim, münaqişənin detallarını və necə həll edildiyini daxil etmək üçün albalı ilə seçilmiş öhdəlik mesajını redaktə etməyə imkan verən açılır. Vim növünü saxlamaq və çıxmaq üçün : wq

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Problem ortaya çıxdıqda, albalı yığımı çox vaxt unudulan son dərəcə faydalı bir texnikadır. Alça yığımının yalnız öhdəliklərinizi kiçik və diqqətinizi cəmlədiyiniz təqdirdə işə yarayacağını qeyd etmək vacibdir, əks halda istənməyən dəyişiklikləri də daxil edəcəksiniz. Bu, Git ilə işləyərkən gündəlik olaraq xatırlamaq üçün vacib bir cəhətdir.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

# Xoş gəlmisiniz!

#### Ssenari 9 - Tarixin yenidən yazılması

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Git'in vacib bir xüsusiyyəti, depo və tarixinizi nə qədər təmiz saxlamağınızdır. Təmiz bir anbarla işləmək və nə baş verdiyini anlamaq daha asandır.

Bu ssenari, dəyişikliklərinizə başlamazdan əvvəl anlaşılan olduqlarını təmin etmək üçün öhdəliklərinizi yenidən qurmaq üçün Rebase istifadə edərək Git tarixçənizi necə yenidən yaza biləcəyinizi əhatə edəcəkdir.

#### Tövsiyə

Yalnız digər insanlarla təkanla paylaşılmayan öhdəlikləri geri qaytarmalısınız. Öhdəliklərin azaldılması, öhdəliklərinin dəyişməsinə səbəb olur və bu da gələcək öhdəliklərin itirilməsinə səbəb ola bilər.

#### Öhdəlik mesajlarında dəyişiklik

Anbarlar tarixçəsini istifadə edərək yenidən yazmaq git rebase -interactive. Yenidən istifadəni interaktiv rejimə keçirtməklə etmək istədiyiniz dəyişikliklərə daha çox nəzarət edə bilərsiniz. İnteraktiv rejimə başladıqdan sonra anbardakı hər bir əməli yerinə yetirmək üçün altı əmr verilir. Açılan redaktoru istifadə edərək, varsayılan olaraq Vim, hər bir öhdəlikdə hansı işləri görmək istədiyinizi müəyyənləşdirirsiniz.

Bu nümunədə öhdəliyi dəyişdirmək istəyirik. Vəziyyəti bu vəziyyətə gətirmək üçün Vim pəncərəsində göstərilən siyahıya əsasən yerinə yetirmək istədiyiniz əməliyyata uyğun olaraq öhdəliyin yanında "seçmək" sözünü dəyişdirməliyik, bu halda "yenidən söz".

Bu nümunədə öhdəlik mesajını dəyişdirmək istəyirik.

#### Başlamaq

Başlamaq üçün istifadə edərək İnteraktiv Rebase rejiminə girməliyik git rebase --interactive --root

#### İnteraktiv rejimi seçin

Vim ilə başlamaq bir az qarışıq ola bilər, mətni düzəltmək üçün əvvəlcə i"daxil etmə rejimi" nə qoyacaqsınız.

İlk siyahıda "siyahının başlanğıcı" mesajında ​​"komit" səhv yazısını düzəltmək istəyirik. Öhdəlik üçün "seçmək" sözünü öhdəlikdə yerinə yetirmək istədiyimiz əmrə uyğun olaraq dəyişdirin, bu halda "yenidən söz".

Yadda saxlamaq və çıxmaq üçün düyməni escbasın :wq. Bu başqa bir Vim redaktor pəncərəsini açacaq.

#### Mesaj dəyişir

Yenidən Vim istifadə edərək, "comit" i "commit" olaraq dəyişdirmək üçün öhdəlik mesajını düzəldin. Vim-i saxladıqdan və çıxdıqdan sonra Git-in çıxışı yerinə yetirməsini dəyişdirdiyini görəcəksiniz. git log --onelineYenilənmiş öhdəlik mesajını görmək üçün istifadə edin .

#### Protip

- kök arqumenti, ilk öhdəlik də daxil olmaqla, depodakı bütün öhdəlikləri yenidən bərpa etməyə imkan verir.

Son öhdəlik mesajını dəyişdirmək üçün daha sürətli bir alternativ git commit --amendVim istifadə edərək dəyişiklik etməkdir.

#### Squash Commits

Yerli mühitdə 8 fərqli öhdəlikdən ibarət bir sıra hazırlanmışdır. O vaxt bu öhdəliklərin mənası var idi, lakin indi yalnız bir öhdəlik olmalıdır. Rebase istifadə etməmiz lazımdır squash birlikdə öhdəliyini.

İstifadədən istifadə edərək git rebase --interactive HEAD~8HEAD-dan son 0-a qədər işləmə aralığımız var.Başlamaq üçün hər şeyin əziləcəyi bir baza öhdəliyinə ehtiyacımız var. Beləliklə, ilk öhdəliyi "seçin" olaraq buraxın, qalanını "balqabaq" olaraq dəyişdirin.

Qurtardıqdan sonra redaktə etmək şansınız olacaq. Varsayılan olaraq, Git törətmə mesajı əvvəlcədən əzilmiş törətmə mesajlarının birləşməsi olacaqdır.

#### Tapşırıq

İnteraktiv Rebase-ə daxil olduqda əvvəlki 8 öhdəliyi istifadə edərək dəyişdirmək istədiyimizi göstərə bilərik git rebase --interactive HEAD~8

Əvvəlki mərhələdə reword istifadə etdik . Burada balqabaqdan istifadə etmək istəyirik . 8 öhdəliyi birinə bükmək istəyirik, əgər bütün öhdəlikləri balqabaq kimi etiketləsəydik, "Əvvəlki öhdəlik olmadan 'balqabaq' alınmaz" səhvini alardıq, çünki hər şeyin içərisinə basıldığı üçün heç bir əsas yoxdur.

Öhdəlikləri əzmək üçün ilk öhdəliyi əsas olaraq qoymalıyıq və aşağıdakı 7-yə balqabaq ilə etiket verin .

#### Mesaj verin

Vimi saxlayarkən və çıxdıqda bizə təzələmədəki 8 törətmə mesajının birləşməsini sadalayan yeni bir Vim pəncərəsi göstərilir.

Öhdəlik mesajını saxladıqdan sonra tarix dəyişdiriləcəkdir. Bunu istifadə edərək görə bilərsinizgit log --oneline

#### Öhdəlikləri yenidən sifariş edin

Öhdəliklərin yenidən sifariş edilməsi, işlənmiş məntiqi sifarişin daha yaxşı bir mənzərəsini yaratmağa kömək edə bilər.

#### Tapşırıq

Son iki işimizi yenidən sifariş etmək istəyirik. HEAD ~ 2 istifadə etmək onları dəyişdirməyə imkan verir.

git rebase --interactive HEAD~2

Vim istifadə edərək, sadəcə xətləri yenidən sıralayın, qənaət edin və çıxın; öhdəliklər sifarişlə uyğunlaşacaq.

#### Bölmə öhdəliyi

Eynilə əmrləri əzməklə olduğu kimi, bəzən dəyişiklikləri diqqət mərkəzində saxlamaq və albalı yığmağı və ya geri qaytarmağı asanlaşdırmaq üçün bölüşdürmək faydalıdır.

Öhdəliklərin bölünməsi iki mərhələli bir prosesdir. Əvvəlcə hansı öhdəliyi bölmək istədiyimizi, sonra da yeni öhdəliklərin necə görünməsini istədiyimizi müəyyənləşdirməliyik.

#### Bölmə öhdəliyini müəyyənləşdirir

Burada əvvəlki öhdəliyi bölmək istəyirik. Rease rejiminə istifadə edərək giririkgit rebase --interactive HEAD~1

Əvvəlki geri qaytarma ilə olduğu kimi, düzəliş üçün tapşırığı dəyişdirməliyik

İndi tarixi interaktiv şəkildə redaktə etmək vəziyyətindəyik. Git bütün dəyişiklikləri qeyd edəcək və son nəticə anbarda tətbiq olunacaq.

#### Bölmə öhdəlikləri

Təsdiqlədikdən sonra öhdəliyi düzəltmək istəyirik , indi tarixi dəyişdirməyə imkan verən bir vəziyyətdəyik.

1. Mövcud bir öhdəliyi bölmək istədiyimiz üçün əvvəlcə onu istifadə edərək aradan qaldırmalıyıq git reset HEAD^.
2. Öhdəlik silindi, lakin sənədlər hələ də mövcuddur. Artıq öhdəlikləri əvvəllər istədiyimiz kimi, iki ayrı hərəkət kimi yerinə yetirə bilərik.

Əmrləri yerinə yetirin:

git add file3.txt  
git commit -m "File 3"  
git add file4.txt  
git commit -m "File 4"

#### Qənaət

Anbarın vəziyyətindən razı qaldıqdan sonra, Git-ə rebasiyaya davam etməsini və anbarı -continue ilə yeniləməsini söyləyirik .

git rebase --continue

Çıxışı və istifadə etdiyi iki yeni vəzifəni görə bilərsiniz git log --oneline

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Tarixçəni yenidən yazma qabiliyyəti, depo tarixçənizi təmiz və dəqiq saxlamağınız üçün faydalıdır. Bu, gələcəkdə dəyişiklik səbəblərini göstərməyə və ya problemləri həll etməyə kömək edəcəkdir.

Bu ssenari, icra etdiyiniz nümunələri və əmrləri nəzərdən keçirə biləcəyiniz albom kitabınıza əlavə edildi.

#### Tövsiyə

Yalnız digər insanlarla təkanla paylaşılmayan öhdəlikləri geri qaytarmalısınız. Öhdəliklərin azaldılması, öhdəliklərinin dəyişməsinə səbəb olur və bu da gələcək öhdəliklərin itirilməsinə səbəb ola bilər.

# Xoş gəlmisiniz!

#### Oyun meydançası

**Çətinlik: Başlanğıc**

**Təxmini vaxt: 10 dəqiqə**

Bu oyun sahəsi Git'i öyrənməyiniz və araşdırmağınız üçün fərdi bir sandboxed mühit təmin edir.

#### Uşaq meydançaları nədir?

Oyun meydançaları, strukturlaşdırılmamış öyrənmə yanaşmasından istifadə edərək oynamağa və araşdırmağa başlamaq üçün sazlanmış bir mühit təqdim edir.

Oyun meydançaları nümunələri sınaqdan keçirmək və sınamaq üçün əladır. Texnologiya haqqında daha çox məlumat əldə etmək üçün [laboratoriyalarımızdan](https://www.katacoda.com/learn) birinə başlayın

#### Faydalı Linklər

Vasitəsilə başlayın git

Anbar istifadəsi yaratmaq git init

Öz Git ssenarilərinizi və demolarınızı yazmaq istəyirsiniz? [Www.katacoda.com/teach saytını](https://www.katacoda.com/teach) ziyarət edin

# Təbrik edirik!

#### Ssenarini başa vurdunuz!

###### **Ssenari reytinqi**

Oyun meydançasını sınadığınız üçün təşəkkür edirik. Daha çox kurs və ssenari [ana səhifəmizdə](https://www.katacoda.com/) mövcuddu