**git add . :**değişiklik yapılan tüm dosyaları ekler, . yerine spesifik olarak isim de belirtilebilir.

**git revert --abort:** Bu komut ile git worktreeye eklediğimiz dosyaları, yani git add . yaptığımız dosyaları sanki add yapmamışız gibi worktreeden temizleme, geri alma işlemini gerçekleştirir. Bu yaptığımız değişiklikleri silmez yalnızca git add komutunu kullanmamışız gibi yapar.

**git status:** klasör içerisindeki dosyaların durumunu gösterir, commit edilmeye hazır olanlar, branche eklenmemiş değişiklikler vs. görülebilir.

**git log:** git dosyasında yapılan değişiklikler vs. log kayırları görünür, kimin hangi commiti yaptığı vs. bilgisi elde edilebilir ve gerektiğinde geri alma işlemleri için bilgi sağlar eğer istenirse git –log –n (1,2,3,…) –n den sonra bir sayı belirterek son yapılan 3 commti veya 5 commiti gibi getirebiliriz.

**git --amend:** bu komut ile en son yapılan commite ek olarak değişiklikler kaydedilebilir, örneğin bir değişiklik yapıldı fakat daha sonra bazı şeylerin eksik kaldığı görüldü ve eksikler giderildi. Eksikler giderildikten sonra yeni bir commit atmak yerine tüm değişiklikler bir önceki committe yapılmış gibi kaydetmek için, git add. Komutundan sonra git commit –amend komutu kullanılarak tek komite kaydedilebilir.  
**git --amend –m :** ile istenirse son yapılan commit in mesajı değiştirilebilir.

**git revert ‘commitID’ :** Kaydedilmiş bir commti git –log komutu ile elde edilebilecek commit has id ile geri alınabilir, bu komut kullanılarak geri alınan bir şeyi tekrar geri alarak hiç geri alınmamış haline getirebiliriz.

**git reset –hard ‘commitId’:** Bu komut ile log kayıtları olmadan yapılan son değişiklikler geri alınamayacak şekilde silinir, kullanımı riskli bir komuttur çünkü belirtilen commitId ye kadar en son yapılan tüm değişiklikler silinir, Örneğin idsi verilen commit sondan 3. İse artık sonuncu commit olur.

**git diff ‘İlk commitId’.. ‘Son commitID’ ‘dosya adı’:** Bu komut ile iki commit karşılaştırılıp hangi değişikliklerin yapıldığı somut bir şekilde görülebilir. Elde edilen sonuçlarda a/’dosyaAdı’ ve b/’dosyaAdı’ şeklinde gösterilen parametreler, iki commiti temsil etmektedir. Eğer komut dosya adı kullanılmadan yazılmış olsaydı o zaman da varsayılan branch üzerinde belirtilen iki commitin etki ettiği dosyalar görülecekti.

**git branch/ git branhc ‘branch adı:** Git ile, ana programa dokunmadan farklı bir dallanma ile yaptığımız değişiklikleri kaydedebilir sonrasında istersek ana programa dahil edebiliriz. Bunu branch komutu ile gerçekleştiririz, branch komutu yalnız başına kullanıldığında projede var olan dalları(branchs) ve hangi dalda bulunduğumuzu gösterir. İhtiyacamıza göre yeni branchler oluşturabiliriz ve yeni branch oluşturmak için git branch ‘branch adı’ komutunu kullanabiliriz.

**git checkout ‘branch adı’:** Bu komut ile üzerinde çalıştığımız branch’i değiştirebiliriz.

**git checkout -b ‘branch adı’:** Bu komut ile yeni bir branch oluştururuz ve direkt üzeinde bulunduğumuz branhc yeni oluşturduğumuz branch olur.

**git branch -D ‘branch adı’:** Bu komut ile branch silme işlemini gerçekleştiririz.

**git stash:** Bu komut üzerinde çalıştığımız branchde yaptığımız son değişiklikleri geçici olarak saklamaya yarar. Örneğin bir commit attık ve daha sonra branchimizde yeni değişiklikler yaptık, fakat sonra fark ettik ki son commitimizde bir şeyleri eksik veya hatalı yapmışız, o hataları düzeltmemiz gerekiyor fakat commit attıktan sonra yeni şeyler yazmıştık bu yazdıklarımızı silecek miyiz? Hayır. Git stash bunun için kullanılıyor, son commit attığımız kısımları düzeltmeden önce git stash komutunu kullandığımızda git bizim için son yaptığımız değişilikleri tutar, bu komuttan sonra dosya son commit attığımız haline döner istediğimiz düzeltmeleri yaptıktan sonra en son kaldığımız yerden devam etmek için ‘git stash pop’ komutunu kullanırız, böylece son attığımız committeki düzeltmelerin üzerine stash komutunu kullanmadan önce eklediğimiz yenilikler de gelir ve kaldığımız yerden devam edebiliriz.

**git stash pop:** Stash listemizdeki en üstte bulunan stashi listeden çıkarır ve ilgili değişilşikleri dosyaya uygular.

**git stash list:** Stash’lerin bir listesini getirir, stash yığıt mantığıyla çalışır.

**git stash apply ‘stashId’:** Stash normalde yığıt mantığıyla çalışır ve en tepeden başlayarak stashleri uygular, ancak stashler arasında spesifik bir tercihimiz varsa en üsttekini değil de diğer stashlerden birini uygulamak istersek stash kullanımı git stash apply ‘stashId’ şeklinde olur.Bu komutu kullanırsak stash listesinden kullandığımız stashi silmez.

**git merge ‘dahil etmek istediğimiz branch’:** Bu komut ile gitte oluşturduğumuz branchleri birleştirebiliriz, dahil etmek istediğimiz branchi dahil edilmesini istediğimiz branch üzerinde git merge ‘dahil edilmesini istediğimiz branch’ şeklinde birleştirebilir. Örneğin header branchini master branchine dahil etmek için, master branchinde git merge header komutunu kullanarak, headerdaki değişiklikleri mastera eklemiş oluruz. Üzerinde bulunduğumuz branch ile dahil etmek istediğimiz branchin ortak commitini buluyor ve o committen en son yapılan commite kadar aradaki commitleri dahil etmek istediğimiz branche(master) ekliyor.

**git merge –squash ‘dahil edilecek branch’:** Bu komut “gir merge ‘dahil etmek istediğimiz branch’” komutu ile aynı işi yapar fakat, bu kullanımdan sonra log kayıtları birleştirilmez tüm birleştirme işlemi ve birkeştirilen branchdeli commitler yeni bir commit ile loglanır.Bunun olması için git merge –squash komutu kullandıktan sonra git bizden son bir commit yazmamızı ister bu commit tüm birleştirme işleminin commitidir. Örneğin header branchini master a dahil ettik ve master’da git log dediğimizde bize headerdaki commit loglarını da getirmez yalnızca git merge –squash komutundan sonra yaptığımız commiti loglar.

**git rebase:** Bu komut merge gibi çalışır fakat yaptığı işlem birleştirme değil sadece dahil edilecek branchdeki commitleri dahil etmek istediğimiz branche ekler fakat bu bir birleştirme işlemi değildir, temelde merge ile aynı işi yapar.

**Conflict:** Confilict git’in çakışma durumunda neyin doğru olduğuna karar veremeyip kararı programcının vermesini istemesi durumudur. Örneğin bir merge işlemi veya fetch/pull işlemi yaptığımızda, bizdeki satırlarla birleştirmek istediğimiz branchde veya remote kaynakta aynı satırlarda bir farklılık varsa yani bizim 11. Satırımız ile karşı taraftaki veya branchdeki 11. Satır farklı ise git hangisinin doğru olduğuna karar vermeiyor ve bu duruma conflict deniyor. Bu durumda hangisinin silinmesini istiyorsak elle silmemiz gerekir. Veya “git merge --abort” komutu ile birleşmeyi geri alıyor.