1. git config --list : bu komut bütün config ayarlarını listeler

2. git config --global user.name = 'SametCavur' : Bu komut ile git'e bir username bırakıyoruz.

3. git config --global user.email = 'example@gmail.com' : Bu komut ile git'e bir mail bırakıyoruz.

4. git config --global --replace-all user.name "Your New Name" : Zaten ekli olan username değiştirme

5. git config --global --replace-all user.email "Your new email" : Zaten ekli olan maili değiştirme

6. git init : cd Dekstop cd file cd other file diyerek working directory(projemizin olduğu klasörün içine kadar gidiyoruz ve daha sonra git init komutunu çalıştırıyoruz.Bu git init o working directory'de bir local repository oluşturuyor.

7. git status : working directory’deki dosyalarımızın durumu hakkında bilgi aldığımız komut.

8. git add . : Burdaki nokta tüm dosyalar demek. Bu komut ise bu klasördeki tüm dosyaları stagging area’ya yani geçiş bölgesine gönder demek.

9. git commit -m ‘Deneme Dosyaları Gönderildi’ : stagging area’ya yani geçiş bölgesine gelmiş dosyaları commit etmek için yani hepsini git’e göndermek için kullandığımız komut. Buradaki -m ‘xxxx’ flagı, dosyaları commit ederken aldığımız not gibi bir şey.

\*\*Üstteki dosyaları commit ettikten sonra yani dosyaları stagging area’ya gönderdikten sonra dosyalarda bir değişiklik yaparsak ve tekrar git status komutuyla kontrol edersek kırmızı kodlar ile modifier yani dosyalarda değişiklik yapıldığını görürüz. Diyelim ki abc.html dosyasında bir değişiklik yaptık üstteki gibi git add abc.html komutunu kullanırsak,değişiklik yaptığımız dosyayı tekrardan stagging area’ya gönderebiliriz.

10. git add -A : yapılan tüm değişiklikleri tekrardan stagging area’ya gönderir.Zaten stagging area’da yada local repoda bulunan ve üzerinde değişiklik yaptığımız dosyaları git status ile kontrol edip sonra git add -A komutu ile tekrardan güncelleyip stagging area’ya ve buradan bir daha git commit -m ‘abc’ komutu ile loca repoya gönderebiliriz.

11. git add abc.html : Bu komut ile üstteki gibi tüm değişiklikleri değil de sadece istediğimiz dosyadaki değişikliğide stagging area’ya gönderebiliriz.

\*\*Önemli: Local Repoya commit edilmiş bir dosya’da değişiklik olursa en başa döner ve tekrardan onu stagging area’ya oradan da local repoya göndermemiz gerekir.

\*\*Şimdiye kadar üstte yaptıklarımız çalışma dizinindeki yani working directory’deki bir dosyayı stagging area’ya yani geçiş bölgesine gönderme, oradan da git’e gönderme ve değişiklikleri git de düzeltme işlemleriydi. Buradan sonraki işlemler ise stagging area’ya gönderilmiş dosyaları tekrardan working directory’e alma işlemi

12. git restore –staged ‘abc.html’ : bu komut stagging area’ya gitmiş olan abc.html dosyasını tekrardan working directory’e getirme komutu

13. git restore abc.html working directory’deki üstünde değişiklik yapılmış olan dosyadaki değişiklikleri geri alıyoruz bu komut ile

14. git show : yaptığımız en son değişikliği gösterir

15. git log : yapılan tüm değişiklikleri gösterir.

16. git log –oneline : yapılan tüm değişiklikleri sade bir şekilde gösterir.

17. git log -p -2 : son 2 değişikliği gösterir.

18. git log --since=30minutes : son 30 dk içerisindeki yapılan değişiklikleri gösterir.

19. git checkout 999999 : bu ise yapılan değişiklikleri silen komut,değişiklik yapmamış gibi geriye alır.Buradaki 999 sayısı ise git log—oneline komutunda gelen logların IDleri.

20. git checkout master: diyelim ki bir üstteki komut ile bir değişikliği geri aldık,geri aldığımız değişikliğide kaldırıp projeyi olduğu hale getirme komutu bu.Yani değişiklik yaptık,değişikliği geri aldık,eski halide olmadı,bir daha yeni haline getirdik.

21. git revert 8888 :diyelim ki 19 nolu nottaki gibi 9999 ID’li işlemi ve 8888 ID’li işlemi sildik.20 nolu nottaki gibi git checkout master komutu ile geri alırsak her 2 işlemde geri alınacak ama biz sadece 8888 ID’li işlemi geri almak istiyorum o zaman git revert 8888 komutunu kullanıp sadece bir değişikliği geri alabiliriz.

22. ESC :x! : VİM’den çıkma komutu

23. git reset –soft 99999: sadece commit alanından kaydettiğimiz tüm verileri siler \*\*Tehlikeli komut\*\*

24. git reset –mixed 99999: stagging area ve commit alanından kaydettiğimiz verileri siler. \*\*Tehlikeli komut\*\*

25. git reset –hard 99999: working directory stagging area ve commit alanından da verileri siler.\*\*Tehlikeli komut\*\*

\*\*Git ignore = cd komutlarıyla working directorymize kadar gidiyoruz ve içeriye touch .gitignore komutunu yazıyoruz. Bu touch .gitignore adında bir dosya oluştur demek. Peki nedir bu git ignore? .gitignore git’e commit etmek istemediğimiz dosyaları tutan dosya diyebiliriz. Mesela log.txt adında bir dosya oluşturduk ve herşeyi commit edeceğiz fakat bu log.txt yi etmeyeceğiz. İşte .gitignore dosyasının içine log.txt yazıyoruz ve bu saatten sonra log.txt yi ne git add . komutuyla stagging area’ya ne de git commit -m ‘aa’ komutuyla git’e gönderebileceğiz. Git’e dosyayı yüklemeden önce daha doğrusu herşeyden önce working directory’e bu gitignore dosyasını koymamız lazım. Eğer bu .gitignore dosyasını koymadan log.txt yi oluşturur ve onu işlemlerden muaf tutmak istersek işimiz biraz zor onuda şöyle yapacağız. İlk önce git rm -r –cached . diyoruz ve tüm dosyaları izlemeden kaldırıyoruz. Daha sonra touch .gitignore ile dosyayı oluşturup ve içine işlemlerden muaf tutmak istediğimiz dosyaları yazıp ,sonra git add . ve git commit -m ‘aa’ komutlarını yazıyoruz.

\*\*Branch olayları : Branch demek çalışma alanı demek gibi bişi diyelim bir proje var ve bu projeyi 2 kişi yapıyoruz. Biri Backendçi diğer Frontendçi git bash’ e şu komutu yazıyoruz. git branch backend bir de git branch frontend , bu isimler kafadan olabilir. Bir tanede master yani ana branch her zaman vardır. Daha sonra diyelim ki git checkout backend yapıyoruz ve artık backendçinin çalışma alanındayız buraya diyelim abc.txt diye bir şey oluşturduk. Artık sadece backendçinin branchine girdiğimizde(git checkout backend) bu abc.txt yi görebiliriz. Şayet git checkout frontend yaparsak o abc.txt’yi göremeyiz. Her iki branchte de görebilmek için master branchte o abc.txt yi oluşturmalıyız.

Branck commands:

git branch xxx : xxx adında bir branch oluştur.

git checkout xxx : xxx branchine gir.

git branch -a : olan tüm branchleri görüntüle.

**Repo:** Projeleri tuttuğumuz depoların her biri. Repository’nin kısaltması. Mesela: <https://github.com/keras-team/keras> bir repo örneğidir.

* **Branch:** Kodların farklı yönlenderdeki geliştirme doğrultuları. Birbirinden ayrılıp ileride birleşen yollar gibi.
* **Commit:** Kodlarda yapılan işlemlerin gönderilmek üzere paketlenmesi.
* **Push:** Paketlenen değişikliklerin sunucuya gönderilmesi.
* **Origin:** Ana veya aktif branhc’e verilen isim. Genelde ilk oluşturulan branch master’dır. İlk origin de dolayısıyla odur.
* **Conflict:** Farklı kişiler tarafından paketlenip suncuya gönderilmeye çalışılan kodların birbiriyle çelişmesi.
* **Merge:** Farklı doğrultuda ilerleyen branch’lerin birleştirilip tek parça şeklinde devam etmesi.
* **Pull:** Başkalarının sunucuya gönderdiği değişiklikleri kendi bilgisayarımızdakine uygulamak için yapılan işlem.
* **Ignore:** Bilgisayarımızda sunucuya göndermek için değişiklik yaptığımızda paketlemeye dahil edilmeyecek dosyaların veya klasörlerin tutulduğu dosya.
* **Clone:** Sunucuda bulunan repoyu aynı şekilde kendi bilgisayarımıza çekmek, klonlamak.
* **Fork:** Başkasına ait bir reponun aynısından kendi hesabımıza klonlamak.