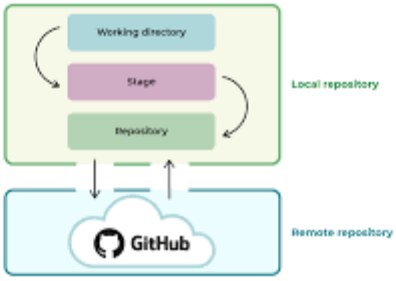
30 Git Komandası

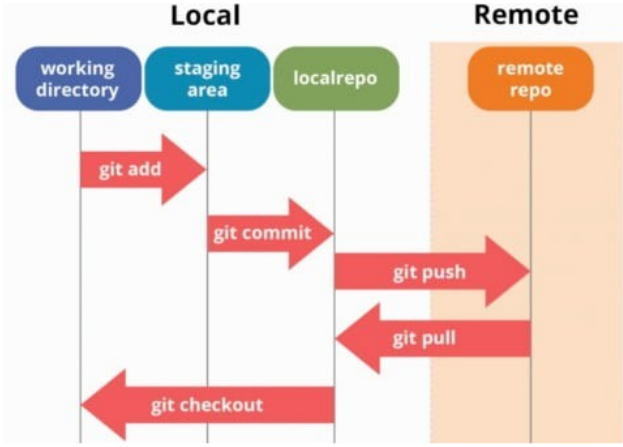
Deməli repository iki cür olur

- **Uzaq -remote repository**
- **Local repository**



Uzaq repository (remote repository) – Git-də layihənizin serverdə saxlanılan nüsxəsidir. Bu, adətən layihənin digər istifadəçilərlə paylaşılması, birgə işlənilməsi üçün istifadə olunur. Uzaq repository ilə yerli repository arasında məlumat mübadiləsi (dəyişiklikləri yükləmək və ya yeniləmək) mümkündür.

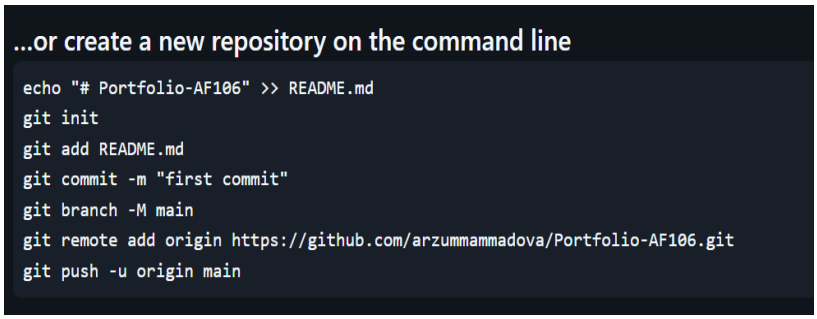
komandalar



1.`git init` - Yeni bir Git repository-i yaratmaq üçün istifadə olunur.

2.`git add <file>` - Dəyişiklikləri `staging area`-ya əlavə etmək üçün istifadə olunur.git add . -Bütün faylı əlavə edir.

3.`git commit -m "<message>"` - `staging area`-dakı dəyişiklikləri Git repository-ə mesajla əlavə edir.



4. **`git branch`** - Mövcud branch-ləri göstərir və yeni branch yaratmaq üçün istifadə olunur.

5. git remote add origin <url>-ayihənizi uzaq bir repository-yə bağlayırsınız

6.**`git push`** – Local dəyişiklikləri uzaq repository-yə göndərir.

7. **`git pull`** – Uzaq repository-dən yeni dəyişiklikləri local repository-yə yükləyir.(bir növ pushun əksini edir)

8. **`git clone <url>`** - Mövcud bir Git repository-i klonlamaq üçün istifadə olunur. Yükləyir yəni

9. **Branch** – Git-də layihənin fərqli qollarını yaratmaq üçün istifadə olunur. Məsələn, yeni bir xüsusiyyət və ya düzəliş üzərində işləmək üçün müstəqil bir qol açırsınız, əsas koda təsir etmədən dəyişikliklər edirsiniz. Hazır olduqdan sonra həmin branch-ı əsas koda birləşdirə (merge) bilərsiniz. Bu, komandada paralel işləməyi asanlaşdırır və dəyişiklikləri təcrid etməyə imkan verir.

Branch Git-də layihənin inkişafını idarə etmək üçün çox vacib bir xüsusiyyətdir. Onun əsas məqsədi, layihənin müxtəlif istiqamətlərində müstəqil olaraq işləyə bilmək, yeni funksiyalar əlavə etmək, xətaları düzəltmək və ya eksperimentlər aparmaq üçün ayrı qollar (bssranches) yaratmaqdır.

Branch-lərin üstünlükləri:

1. **Müstəqil işləmək**: Layihədə yeni bir xüsusiyyət (feature) üzərində işləmək üçün yeni bir branch yarada bilərsiniz və əsas koda zərər vermədən müstəqil dəyişikliklər edə bilərsiniz.
2. **Dəyişiklikləri təcrid etmək**: Hər bir branch layihənin bir fərqli versiyasını təmsil edir. Bu da, xüsusiyyətlər, düzəlişlər və təcrübələr kimi işləri təcrid etməyə imkan verir.
3. **Geri dönüş imkanları**: Əgər yeni xüsusiyyət və ya düzəlişlər branch-da işlənildikdən sonra problem yaranarsa, əsas (master/main) branch-a toxunulmadığı üçün asanlıqla geriyə dönmək mümkündür.
4. **Ekip işi üçün uyğundur**: Müxtəlif inkişafçılar layihənin müxtəlif hissələri üzərində eyni vaxtda müstəqil işləyə bilirlər. Hər bir inkişafçı öz branch-ında işləyib, dəyişikliklərini sonra birləşdirə bilər.
5. **Test etmək üçün mükəmməl mühit**: Branch-lərdə yeni xüsusiyyətləri və ya təkmilləşdirmələri test edib, tam əmin olduqdan sonra əsas branch-a birləşdirə bilərsiniz.

Məsələn:

Təsəvvür edin, əsas layihə **`main`** branch-da işləyir və siz yeni bir xüsusiyyət üzərində işləmək istəyirsiniz. Bunun üçün **`feature-new`** adlı bir branch yaradırsınız. Bu branch-da işləyib tam hazır olduqdan sonra onu **`main`** branch-a birləşdirirsiniz. Əgər bir problem olarsa, sadəcə bu branch-ı silib, layihənizin əsas versiyasına zərər vermədən işləməyə davam edə bilərsiniz.

Branch-lərin istifadəsi layihənin kod keyfiyyətini qorumağa, test etməyə və daha asanlıqla idarə etməyə kömək edir.

10. **`git clean -f`** - İzlənilməyən faylları silir.

11. **`git reset --hard <commit>`** - Commit-dən sonrakı bütün dəyişiklikləri ləğv edir.

12. **`git revert <commit>`** - Müəyyən bir commit-in dəyişikliklərini geri alır.

13. **`git status`** - Repository-nin cari vəziyyətini, hansı faylların dəyişdiyini, `staging area`-ya əlavə olunduğunu göstərir.

14. **`git diff`** - Dəyişiklikləri göstərir.

15. **`git diff <source_branch> <target_branch>`** - İki branch arasındakı fərqləri göstərir.

16. **`git log`** - Commit tarixçəsini göstərir.

17. **`git fetch`** - Uzaq repository-dən məlumatları gətirir, amma onları yerli branch-ə birləşdirmir.

Git pull amma yükləyir

18. **`git log --oneline`** - Commit tarixçəsini qısa şəkildə göstərir.
19. **`git commit --amend`** - Ən son commit-i redaktə etmək üçün istifadə olunur.

20. **`git remote add <name> <url>`** - Yeni bir uzaq repository əlavə edir.

21. **`git cherry-pick <commit>`** - Müəyyən bir commit-in dəyişikliklərini seçib, cari branch-ə əlavə edir.

22. **`git bisect`** - Problemi yaradan commit-i tapmaq üçün istifadə olunur.

23. **`git archive`** - Repository-nin xüsusi bir hissəsini arxiv formatında çıxarmaq üçün istifadə olunur, adətən layihənin bir snapshotunu yaratmaq üçün.

24. **`git show <commit>`** - Müəyyən bir commit-in detallarını, dəyişikliklərini və metadata-sını göstərir.

25. **`git rm <file>`** - Faylı Git repository-dən silir və onu `staging area`-ya əlavə edir.

26. **`git clean -f`** - İzlənilməyən faylları silir.

27. **`git stash`** - Cari dəyişiklikləri müvəqqəti olaraq saxlamaq üçün istifadə olunur.

28. **`git stash apply`** - Sonuncu stash edilmiş dəyişiklikləri geri qaytarır.

29. **`git tag <tagname>`** - Layihəyə bir tag əlavə edir.

30. **`git config --global user.name "<name>"`** - Git istifadəçi adını konfiqurasiya edir.