https://www.edureka.co/blog/interview-questions/git-interview-questions/

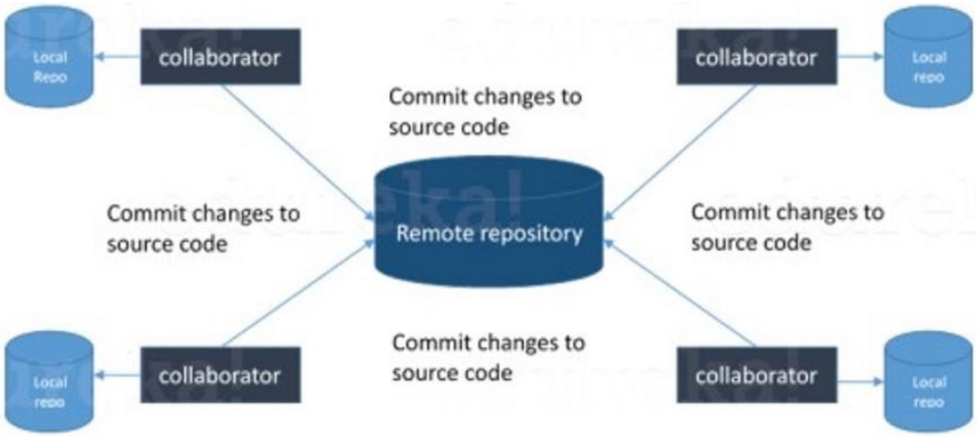
1. Mi a különbség a Git és az SVN között?

Git	SVN
A Git egy decentralizált verziókezelő eszköz	Az SVN egy központosított verziókezelő eszköz
A Version Control eszközök 3. generációjához tartozik	A Version Control eszközök 2. generációjába tartozik
Az ügyfelek teljes repository-kat klónozhatnak helyi rendszereiken	A verzióelőzmények egy kiszolgálóoldali tárolóban vannak tárolva
A commit-okoffline állapotban is lehetségesek	Csak online commit-ok megengedettek
A push / pull műveletek gyorsabbak	A push / pull műveletek lassabbak
A munkákat automatikusan megosztja a commit	Semmit sem osztanak meg automatikusan

2. Mi az a Git?

Azt javaslom, hogy próbálkozzon ezzel a kérdéssel úgy, hogy először elmondja a git architektúráját, amint az az alábbi ábrán látható, csak próbálja meg elmagyarázni a diagramot:

- A Git egy elosztott verziókezelő rendszer (DVCS). Segítségével nyomon követheti a fájlban végrehajtott módosításokat, és visszatérhet a kívánt módosításokhoz.
- Ez egy elosztott architektúra, amely számos előnyt nyújt más verziókezelő rendszerekhez (VCS), például az SVN-hez képest. Az egyik fő előny, hogy nem egy központi szerverre támaszkodik a projekt fájljainak összes verziójának tárolására.
- Ehelyett minden fejlesztő „klónozza” az ábrán bemutatott repository egy példányát a „Helyi repository-val”, és a projekt teljes előzménye elérhető a merevlemezén. Tehát amikor kiszolgáló leáll, csak a csapattársa egyik helyi Git-tárházára van szüksége a helyreállításhoz.
- Van egy központi felhőtár, ahol a fejlesztők változtatásokat hajthatnak végre, és megoszthatják azokat más csapattársaikkal.



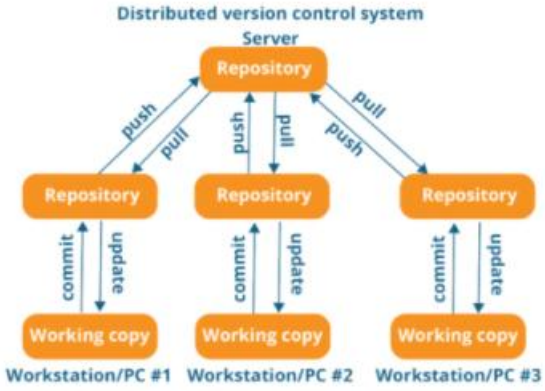
3. Mi az elosztott VCS?

Ezek azok a rendszerek, amelyek nem támaszkodnak egy központi szerverre a projektfájl és annak összes verziójának tárolásához.

Az elosztott VCS-ben minden közreműködő beszerezheti a fő repository helyi másolatát vagy „klónját”.

Amint a fenti ábrán látható, minden programozó fenntarthat egy helyi adattárat, amely valójában a merevlemezen található központi repository másolata vagy klónja. Minden gond nélkül commit-olhatja és frissíthetik a helyi repository-t.

A „pull” nevű művelettel frissíthetik a helyi repository-t a központi szerverről származó új adatokkal, és a „pull” művelet hatással van a helyi repository fő repository változásaira.



4. Mi a különbség Git és Github között?

A Git egy elosztott verziókezelő rendszer, amelyet a forráskód változásainak nyomon követésére használnak a szoftverfejlesztés során. Segíti a programozók munkájának összehangolását, de bármely fájlkészlet változásainak nyomon követésére használható. A Git fő célkitűzései a sebesség, az adatok integritása és az elosztott, nem lineáris munkafolyamatok támogatása.

A GitHub egy Git-repository-tárhelyszolgáltatás, ráadásul számos saját funkcióval rendelkezik. A GitHub webalapú grafikus felületet biztosít. Emellett hozzáférés-vezérlést és számos együttműködési funkciót, alapvető feladatkezelő eszközöket biztosít minden projekthez.

5. Milyen előnyei vannak a verziókezelő rendszer használatának?

A verziókezelő rendszer (VCS) használatával a csapat minden tagja bármikor szabadon dolgozhat bármilyen fájlon. A VCS rugalmasságot kínál az összes változás közös verzióvá egyesítéséhez.

Az összes előző verzió és változat szépen be van csomagolva a VCS-be. Bármelyik verziót igényelheti igényei szerint, és kéznél lesz egy pillanatkép a teljes projektről.

A projekt új verziójának mentésekor a VCS megköveteli, hogy rövid leírást adjon a végrehajtott módosításokról. Ezenkívül láthatja, hogy a fájl tartalma milyen változtatásokat hajt végre. Ez segít megtudni, hogy milyen változtatásokat hajtottak végre a projektben és ki hajtotta végre.

Az olyan elosztott VCS, mint a Git, lehetővé teszi a csapat összes tagjának, hogy teljes előzményekkel rendelkezzen a projektről, így ha a központi szerverben meghibásodás tapasztalható, használhatja csapattársa bármelyik helyi Git-adattárát.

6. Milyen nyelvet használ a Git?

Ahelyett, hogy csak megmondaná a nyelv nevét, meg kell mondania a használat okát is. Azt javaslom, hogy válaszoljon erre:

Git 'C' nyelvet használ. A GIT gyors, és a „C" nyelv ezt lehetővé teszi azáltal, hogy csökkenti a magas szintű nyelvekhez tartozó futási idők általános költségeit.

7. Említse meg a különféle Git-repository hosting funkciókat.

- Github
- Gitlab
- Bitbucket
- SourceForge
- GitEnterprise

8. Mi az a commit üzenet?

A commit üzenet megírásához használt parancs a „git commit -a".

Most magyarázza el az -a jelzőt úgy, hogy a -a parancssorban azt írja, hogy a git utasítja az összes módosított nyomon követett fájl új tartalmát. Emellett említsd meg, hogy a „git add <file>" szót használhatod a git commit -a előtt, ha új fájlokat kell először commit-olni.

9. Hogyan lehet kijavítani a megsértett commit-ot?

A megszakadt commit kijavításához használja a „git committ --amend” parancsot. A parancs futtatásakor a szerkesztőben kijavíthatja a meghibásodott commit üzenetet.

10. Mi az a repository a Gitben?

A Git repository egy olyan hely, ahol a Git tárolja az összes fájlt. A Git a fájlokat a helyi vagy a távoli repository-ban tárolhatja.

11. Hogyan hozhat létre repository-t a Git-ben?

Valószínűleg ez a leggyakrabban feltett kérdés, és a válasz erre nagyon egyszerű.

Repository létrehozásához hozzon létre egy könyvtárat a projekthez, ha az még nem létezik, majd futtassa a „git init” parancsot. A parancs futtatásával a .git könyvtár létrehozásra kerül a projekt könyvtárban.

12. Mi a „csupasz tárház” a Gitben?

A Gitben található „csupasz” adattár információkat tartalmaz a verziókezelésről és nem tartalmaz működő fájlokat (nincs fa), és nem tartalmazza a speciális .git alkönyvtárat. Ehelyett a .git alkönyvtár összes tartalmát közvetlenül magában a főkönyvtárban tartalmazza, míg a munkakönyvtár a következőkből áll:

- .Git alkönyvtár a repository összes Git-tel kapcsolatos előzményével.
- Működő fa, vagy a projektfájlok másolatai.

13. Mi a „konfliktus” a gitben?

A Git önállóan képes kezelni a merge-ket az automatikus merge funkciók használatával. Konfliktus merül fel, ha két külön ág szerkesztett egy fájlt ugyanazon soron, vagy ha az egyik ágban töröltek egy fájlt, de a másikban szerkesztettek. Konfliktusok akkor fordulnak elő leginkább, ha csapatkörnyezetben dolgoznak.

14. Hogyan használják a git instawebet?

A „git instaweb” arra szolgál, hogy automatikusan irányítson egy webböngészőt és futtasson egy webszervert egy felülettel a helyi repository-ba.

15. Mi az a git is-tree?

A „git is-tree” egy fa objektumot jelent, beleértve az egyes elemek módját és nevét, valamint a blob vagy a fa SHA-1 értékét.

16. Nevezzen meg néhány Git parancsot, és magyarázza el azok használatát.

Az alábbiakban néhány alapvető Git parancs található:

Command	Function
`git rm [file]`	törli a fájlt a munkakönyvtárból
`git log`	sorolja fel az aktuális ág verzióelőzményeit.
`git show [commit]`	megmutatja a megadott commit metaadatait és tartalmi változásait.
`git tag [commitID]`	címkék adására szolgál a megadott commit-hoz.

`git checkout [branch name]`
 	
`git checkout -b [branch name]`
	egyik ágról a másikra váltani.

17. Hogyan oldható meg egy konfliktus Gitben?

- Határozza meg a konfliktust okozó fájlokat.
- Hajtsa végre a szükséges módosításokat a fájlokban, hogy ne fordulhasson elő ismét konfliktus.
- Adja hozzá ezeket a fájlokat a git add paranccsal.
- Végül a megváltozott fájlt a git commit paranccsal kell commit-olni.

18. A Git-ben hogyan lehet visszavonni egy olyan commit-ot, amelyet már push-oltak és nyilvánosságra hoztak?

Kétféle módon lehet kezelni ezt a kérdést, és győződjön meg róla, hogy mindkettőt tartalmazza-e, mert az alábbi lehetőségek bármelyike alkalmazható a helyzettől függően:

Távolítsa el vagy javítsa ki a hibás fájlt egy új commit-ban, majd push-olja a távoli repository-ba. Ez a legkézenfekvőbb módszer a hiba kijavítására. Miután elvégezte a szükséges változtatásokat a fájlban, akkor a távoli repository-ba vezesse be a következő paranccsal: git commit -m “commit message”

Ezenkívül létrehozhat egy új commit-ot, amely visszavonja a rossz commit-ban végrehajtott összes módosítást. Ehhez használja a parancsot

```
git revert <name of bad commit>
```

20. Mi a különbség a git pull és a git fetch között?

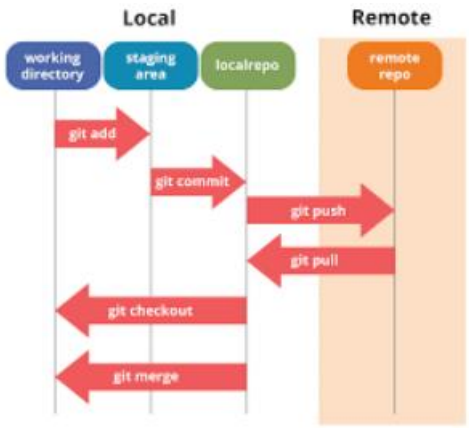
A Git pull parancs új módosításokat hajt végre vagy végrehajt egy adott branch-ről a központi repository-ba, és frissíti a target branch-et a helyi repository-ban.

A Git fetch-et is ugyanarra a célra használják, de kissé másképp működik. Amikor végrehajt egy git fetch parancsot, az összes új commit-ot pull-olja a kívánt repository-ból, és egy új branch-ben tárolja a helyi repository-ban. Ha ezeket a változásokat a target branch-ban szeretné tükrözni, a git fetch parancsot git merge-el kell követni. A target branch csak a target branch és a módosított branch összevonása után frissül. Ne feledje az alábbi egyenletet, hogy megkönnyítse az Ön számára:

Git pull = git fetch + git merge

21. Mi az „átmeneti terület (Staging area)" vagy „index" Gitben?

A commit teljesítése előtt formázható és felülvizsgálható egy „Staging Area" vagy „Index" néven ismert köztes területen. A diagram alapján nyilvánvaló, hogy minden változtatást először ellenőriznek az átmeneti területen, amelyet „stage file" neveztem el, majd ezt a változást commit-oljuk a repository felé.



22. Milyen munka áll helyre a törölt ág helyreállításakor?

Az stash-ed és a stash index-listába mentett fájlokat visszaállítja. A nem követett (untracked) fájlok elvesznek. Célszerű a munkáját mindig stage-be állítani és commit-olni, vagy elrejteni (stash).

Ha be akarja szerezni egy adott ág vagy tag napló hivatkozásait, akkor futtassa a „git reflog <ref_name>" parancsot.

23. Mi a git stash?

Gyakran, amikor a projekt egy részén dolgozott, a dolgok rendetlen állapotban vannak, és egy ideig branch-et akar váltani, hogy mással dolgozzon. A probléma az, hogy nem akar félkész munkát commit-olnicsak később térhet vissza erre a pontra. A válasz erre a kérdésre a Git stash.

A Stashing a munkakönyvtárat, vagyis a módosított nyomon követett fájlokat és a lépcsőzetes módosításokat veszi fel, és elmenti egy sor befejezetlen változtatásra, amelyeket bármikor újra alkalmazhat.

24. Mi a „ git stash apply" funkciója?

Ha továbbra is ott akar dolgozni, ahol abbahagyta a munkáját, akkor a „git stash apply" paranccsal visszahozza a mentett módosításokat az aktuális munkakönyvtárba.

25. Mi a különbség a „git diff" és a „git status" között?

A „git diff" a változásokat ábrázolja az commit-ok között, a commit és a working tree között stb. Míg a „git status" megmutatja a különbséget a munkakönyvtár és az index között, hasznos a git átfogóbb megértésében. A „git diff" hasonló a „git status" -hoz, az egyetlen különbség az, hogy megmutatja a különbségeket a különféle commit-ok, valamint a munkakönyvtár és az index között.

26. Mi a különbség a „git remote" és a „git clone" között?

A „git remote add" létrehoz egy bejegyzést a git config-ban, amely megadja az adott URL nevét, míg a „git clone" új git repository-t hoz létre az URL-n található meglévő másolásával

27. Mi a git stash drop?

A Git 'stash drop' paranccsal eltávolíthatja az elrejtett (stashed) elemet. Alapértelmezés szerint eltávolítja az utoljára hozzáadott rejtett (stashed) elemet, és egy adott elemet is eltávolíthat, ha argumentumként adja meg.

Most mondjon példát.

Ha el akar távolítani egy adott rejtett (stashed) elemet az elrejtett elemek listájáról, használhatja az alábbi parancsokat:

git stash list: Megjeleníti az elrejtett elemek listáját, például:

```
stash@{0}: WIP on master: 049d078 added the index file
stash@{1}: WIP on master: c264051 Revert "added file_size"
stash@{2}: WIP on master: 21d80a5 added number to log
```

If you want to remove an item named stash@{0} use command **git stash drop stash@{0}**.

28. Hogyan találja meg azoknak a fájloknak a listáját, amelyek megváltoztak egy adott commit-nál?

Erre a válaszra ahelyett, hogy csak a parancsot mondaná el, magyarázza el, hogy pontosan mit fog tenni ez a parancs.

Az adott paranccsal megváltozott fájlok listáját a commit-ban használja az alábbi parancsot:

```
git diff-tree -r {hash}
```

Tekintettel a commit hash-re, ez felsorolja az összes olyan fájlt, amelyet megváltoztattak vagy hozzáadtak a commit-hoz. A -r jelző a parancslistát egyedi fájlokká teszi, ahelyett, hogy csak gyökérkönyvtár nevekbe bontaná őket.

Felveheti az alább említett pontot is, bár teljesen opcionális, de segít lenyűgözni az interjúztatót.

A kimenet tartalmaz még néhány extra információt, amelyeket két flag beillesztésével könnyen meg lehet szüntetni:

```
git diff-tree --no-commit-id --name-only -r {hash}
```

–no-commit-id elnyomja a commit hash megjelenését a kimenetben

–name-only csak a fájlneveket írja ki, a path helyett.

29. Mi a „git config” funkciója?

Git a felhasználónévvel társítja a commit-ot egy identitáshoz. A git config paranccsal megváltoztathatja a Git konfigurációt, beleértve a felhasználónevét is.

Most magyarázza el egy példával.

Tegyük fel, hogy meg akar adni egy felhasználónevet és e-mail azonosítót a commit társításához egy identitáshoz, hogy megtudhassa, ki adott commit-ot. Ehhez a következőket fogom használni:

`git config –global user.name “Your Name”`**:** This command will add a username.
`git config –global user.email “Your E-mail Address”`**:** This command will add an email id.

30. Mit tartalmaz egy commit objektum?

A Commit objektum a következő összetevőket tartalmazza, meg kell említenie az alábbiakban bemutatott mindhárom pontot:

- Fájlkészlet, amely a projekt állapotát mutatja egy adott időpontban
- Hivatkozás szülő objektumokra

- SHA-1 név, 40 karakter hosszúságú karakterlánc, amely egyedileg azonosítja a commit objektumot

31. Írja le az elágazási (branching) stratégiákat, amelyeket használt.

- Funkció (feature)-elágazás - A szolgáltatás-elágazási modell egy adott szolgáltatás összes változását megtartja az elágazáson belül. Amikor a funkciót teljesen tesztelték és automatizált tesztekkel validálták, az ágat egyesítik a masterbe.
- Feladat elágazás - Ebben a modellben minden feladatot a saját ágán hajtanak végre, az ág nevében szereplő feladat kulccsal. Könnyen belátható, melyik kód melyik feladatot valósítja meg, csak a branch kulcsában keresse meg a feladat kulcsát.
- Kibocsátási (release) elágazás - Miután a fejlesztési elágazás elegendő funkciót szerzett egy kiadáshoz, klónozhatja ezt az elágazást, hogy release elágazást képezzen. Ennek az ágnak a létrehozása elindítja a következő kiadási ciklust, így ezen pont után nem lehet új funkciókat hozzáadni, csak hibajavításokat, dokumentáció-generálást és egyéb kiadásorientált feladatokat kell elvégezni ebben az ágban. Miután készen áll a deploy-ra, a kiadás beolvasztásra kerül a masterbe, és meg van jelölve egy verziószámmal. Ezenkívül vissza kell illeszteni a fejlesztési ágba, amely előreléphetett a kiadás megkezdése óta.

Végül mondd el nekik, hogy az elágazási stratégiák szervezetenként változnak, mint az alapvető elágazási műveleteket, mint delete, merge, checking out a branch stb.

32. Magyarázza el a forking munkafolyamat előnyeit!

Alapvető különbség van az forking munkafolyamat és más népszerű git munkafolyamat között. Ahelyett, hogy egyetlen szerveroldalt használna „központi” kódbázisként, minden fejlesztőnek megadja a saját szerveroldali repository-ját. A Forking Workflow általában nyilvános nyílt forráskódú projektekben látható.

A Forking Workflow kulcsfontosságú előnye, hogy a hozzájárulások integrálhatók anélkül, hogy mindenkire szükségük lenne egyetlen központi repository-hoz való hozzáféréshez, amely tiszta projektelőzményekhez vezet. A fejlesztők saját szerveroldali tárhelyeikre léphetnek, de csak a projektfenntartó léphet a hivatalos adattárba.

Ha a fejlesztők készen állnak egy helyi commit közzétételére, akkor a saját nyilvános adattárukra küldik az commit-ot, nem pedig a hivatalosba. Ezek után egy pull request-et keresnek a main repository-val, amely a projekt fenntartójának tudatja, hogy a frissítés készen áll az integrálásra.

33. Honnan fogja tudni a Git-ben, ha egy branch-et már beolvasztottak a masterbe?

Használhatja az alábbi parancsokat, hogy megtudja, hogy egy branch-et beolvasztottak-e a masterbe vagy sem.

- git branch --merged - Felsorolja azokat az ágakat, amelyek beolvadtak a jelenlegi ágba.
- git branch --no-merged - Azokat az ágakat sorolja fel, amelyeket nem egyesítettek.

34. Miért kívánatos egy további commit létrehozása a meglévő commit módosítása helyett?

Ennek pár oka van -

- A módosító művelet elpusztítja azt az állapotot, amelyet korábban egy commit-ban mentettek el. Ha csak a commit üzenetet változtatják meg, akkor ez nem probléma. De ha a tartalmat módosítják, akkor továbbra is nagyobb az esély egy fontos dolog kiküszöbölésére.
- A „ git commit- amend" használata egy kis commit növekedését eredményezheti, és nem kapcsolódó változásokat eredményezhet.

35. Mit tartalmaznak a „ hooks" a Git-ben?

Ez a könyvtár shell parancsfájlokból áll, amelyek aktiválódnak, ha a megfelelő Git parancsokat futtatja.

36. Hogyan adná vissza a Gitben egy olyan commit-ot, amelyet most push-oltak és tettek nyilvánossá?

Egy vagy több commit visszavonható a git revert használatával. Ez a parancs valódi értelemben új commit-ot hoz létre javításokkal, amelyek megsemmisítik az egyes commit-okban bevezetett változásokat. Ha abban az esetben, ha a visszavonásra szoruló commit-ot már közzétették, vagy ha a repository előzményeinek módosítása nem lehetséges, akkor ilyen esetekben a git revert parancs használható a visszavonások visszavonására. Ha a következő parancsot futtatja, akkor az visszavonja az utolsó két parancsot:

```
git revert HEAD~2..HEAD
```

Alternatív megoldásként mindig van lehetőség arra, hogy ellenőrizze egy adott múltbeli commit állapotát, és új commit-ot készítsen.

37. Hogyan lehet eltávolítani egy fájlt a gitből anélkül, hogy eltávolítanánk a fájlrendszeréből?

Óvatosnak kell lennie a git hozzáadása során, különben olyan fájlokat adhat hozzá, amelyeket nem akart commit-olni. A git rm azonban eltávolítja mind az átmeneti területről (index), mind a fájlrendszeréről (working tree), ami nem biztos, hogy a kívánt.

Ehelyett használja a git reset parancsot:

```
git reset filename      # or
```

```
echo filename >> .gitingore # add it to .gitignore to avoid re-adding it
```

Ez azt jelenti, hogy a git reset <paths> ellentétes a git add <paths> -al.

38. Meg tudja magyarázni a Gitflow munkafolyamatot?

A projekt történetének rögzítéséhez a Gitflow munkafolyamat két párhuzamos, hosszú távú ágat alkalmaz - master és develop:

- Master - ez az ág mindig készen áll a LIVE-on való kiadásra, mindent teljesen tesztelve és jóváhagyva (gyártásra kész).
- Gyorsjavítás (Hotfix)- ezeket az ágakat a gyártási kiadások gyors javításához használják. Ezek az ágak nagyon hasonlítanak a release ágakhoz és a feature ágakhoz, kivéve, hogy a development helyett a masteren alapulnak.
- Fejlesztés (Development)- ez az az ág, amelyhez az összes jellemző ág összeolvad, és ahol az összes tesztet elvégzik. Csak akkor lehet egyesíteni a masterrel, ha mindent alaposan ellenőriztünk és kijavítottunk.
- Feature - minden új funkciónak a saját ágában kell lennie, amelyet a fejlesztési ághoz lehet push-olni, mint szülőjüket.

39. Mondja el a különbséget a HEAD, a working tree és az index között Git-ben.

A working tree/working directory/workspace a (forrás) fájlok könyvtárfája, amelyeket megtekinthet és szerkeszthet.

Az index/staging area egyetlen, nagy, bináris fájl a <baseOfRepo> /.git/index fájlban, amely felsorolja az aktuális ág összes fájlját, SHA-1 ellenőrző összegeit, időbélyegeit és a fájl nevét - ez nem egy másik könyvtár amely a fájlok másolatát tartalmazza.

A HEAD a jelenleg kijelölt branch-ben az utolsó commit-ra utal.

40. Mi a Git fork? Mi a különbség a fork, az branch és a klón között?

A fork a tároló másolata. Normál esetben elágazik egy repository, hogy szabadon kísérletezhessen a változtatásokkal anélkül, hogy befolyásolná az eredeti projektet. Leggyakrabban a fork-okat arra használják, hogy javaslatot tegyenek mások projektjeire, vagy hogy mások projektjét használják kiindulópontként saját ötletükhöz.

A git klónozás azt jelenti, hogy rámutatunk egy meglévő repository-ra, és másolatot készítünk róla egy új könyvtárban, valamilyen más helyen. Az eredeti repository megtalálható a helyi fájlrendszeren vagy a távoli gép által elérhető, támogatott protokollokon. A git clone paranccsal egy meglévő Git-repository másolatát lehet létrehozni.

Nagyon egyszerű szavakkal, a git ágak egyedi projektek a git adattárban. A táron belüli különféle ágak teljesen más fájlokkal és mappákkal rendelkezhetnek, vagy ugyanazok lehetnek, kivéve a fájl néhány kódsorát.

41. Milyen módon hivatkozhat egy commit-ra?

A Git-ben minden commit-nak egyedi hash-ja van. Ezeket a kivonatokat használják a megfelelő végrehajtások azonosítására különböző esetekben, például a kód egy adott állapotának a git checkout {hash} paranccsal történő megpróbálására.

Ezzel párhuzamosan Git számos alias-t vezet be bizonyos commit-okhoz, amelyeket ref-nek neveznek. Ezenkívül minden, a repository-ban létrehozott tag hatékonyan referenciává válik, és pontosan ezért használhatja a tag-eket ahelyett, hogy hash-eket végezne a különféle git parancsokban. A Git számos speciális alias-t is fenntart, amelyek a repository állapota alapján megváltoznak, például HEAD, FETCH_HEAD, MERGE_HEAD stb.

A Gitben a commit-okat egymáshoz viszonyítva lehet hivatkozni. Merge esetén, ahol a commit-nakkét szülője van, a ^ segítségével kiválasztható a két szülő közül az egyik, például a HEAD ^ 2 segítségével követhető a második szülő.

Végül a refspeceket a helyi és a távoli ágak együttes feltérképezésére használják. Ezeket azonban fel lehet használni olyan távoli elágazásokon elhelyezkedő commit-okra is, amelyek lehetővé teszik az irányítást és a helyi git környezetből történő manipulálást.

42. Mi a különbség a rebase és a merge között a Git-ben?

A Gitben a rebase paranccsal integrálhatók az egyik ágból a másikba végrehajtott változások. Ez az „merge” parancs alternatívája. A különbség a rebase és a merge között az, hogy a rebase újraírja az commit előzményeit annak érdekében, hogy egyenes, lineáris sorozatot hozzon létre.

A merge Git módja, hogy a fork történelmet újra összeállítsa. A git merge parancs segít a git-ág által létrehozott független fejlődési vonalak felvételében és egyetlen ágba integrálásában.

43. Magyarázza el a különbséget a reverting és a resetting között.

A Git reset egy erőteljes parancs, amelyet a Git-repository állapotának helyi változásainak visszavonására használnak. A Git reset a „Git három fája (The Three Trees of Git)”, a Commit History (HEAD), a Staging Index és a Working Directory segítségével működik.

A Revert parancs a Gitben új commit-ot hoz létre, amely visszavonja az előző commit változásait. Ez a parancs új előzményeket ad a projekthez. Nem módosítja a meglévő előzményeket.

44. Mi az a git cherry-pick?

A git cherry-pick parancsot általában arra használják, hogy egy adott repository egy ágából származó különféle commit-ot másik ágra vezesse be. Egy másik gyakori használat, ha a karbantartási ágtól a fejlesztési ágig előre vagy hátra . Ez ellentétben áll más módszerekkel, például a merg-el és rebase-el, amely általában sok commit-ot alkalmaz egy másik ágra.

```
git cherry-pick <commit-hash>
```

45. Hogyan találja meg azokat a fájlokat, amelyek megváltoztak egy adott commit során?

```
git diff-tree -r {hash}
```

Tekintettel a commit hash-re, ez felsorolja az összes olyan fájlt, amelyet megváltoztattak vagy hozzáadtak a commit-hoz. A -r jelző a parancslistát egyedi fájlokká teszi, ahelyett, hogy csak gyökérkönyvtár nevekbe bontaná őket.

A kimenet tartalmaz még néhány további információt, amelyeket könnyen el lehet nyomni néhány flag hozzáadásával:

```
git diff-tree --no-commit-id --name-only -r {hash}
```

–no-commit-id: elnyomja commit hash megjelenését a kimenetben

–name-only: csak a fájlneveket írja ki, az útvonaluk helyett