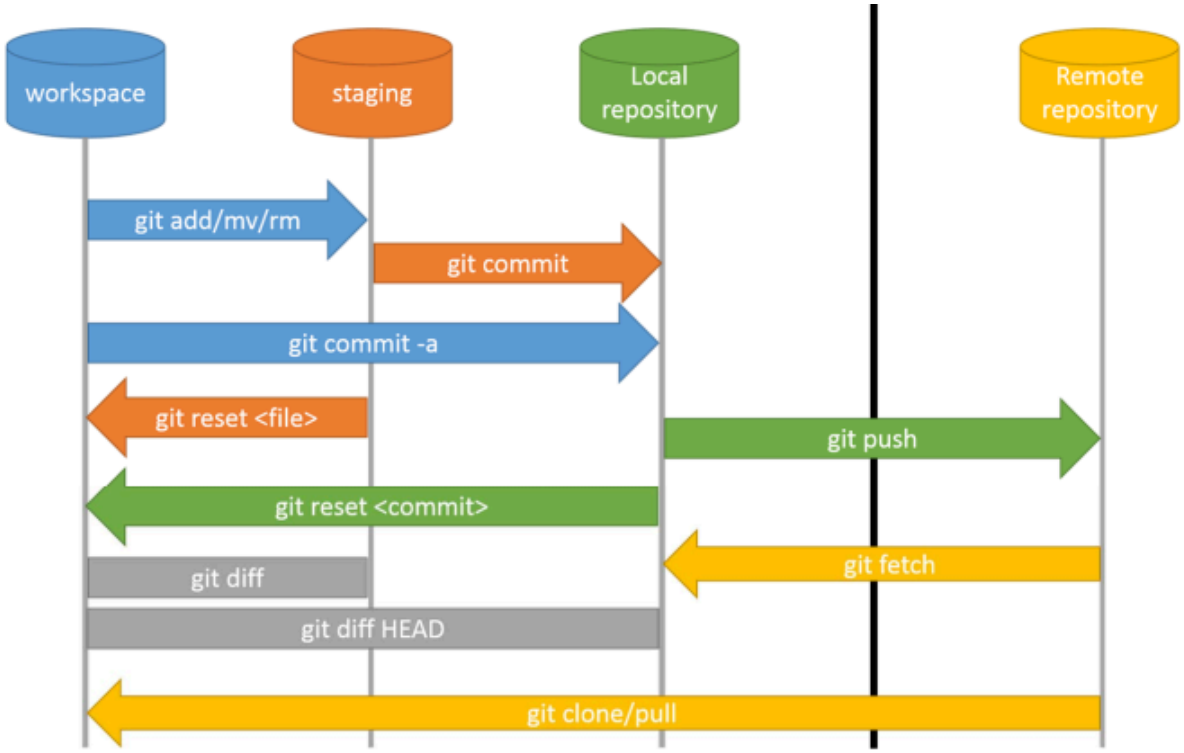
Git

Rövid útmutató – működésről és parancsokról

—

Git működése



A fenti ábra a Git működésének alapvető folyamatát szemlélteti. Négy különböző területet mutat be: **workspace**, **staging**, **local repository** (helyi tárhely) és **remote repository** (távoli tárhely). Ezek közötti mozgásokat jelöli a diagram különböző parancsokkal. Lássuk, hogyan működnek ezek a lépések:

1. Workspace

Ez az a könyvtár, ahol a fejlesztő ténylegesen dolgozik a fájlokon. Itt készülnek a változtatások.

- **`git add <fájl>`** vagy **`git mv <fájl>`** vagy **`git rm <fájl>`**: Ezek a parancsok a módosított, átnevezett, vagy törölt fájlokat a **staging** területre küldik. Ez egy úgymond előkészítő terület, ahol a fájlok commitolásra várnak.

2. Staging

A staging tartalmazza a fájlokat, amelyek készen állnak a commitra. Itt gyűjtöd össze azokat a változtatásokat, amelyeket commitolni szeretnél.

- **`git commit`**: Ez a parancs commitolja a staging területen található változtatásokat a **local repository**-ba. Egy commit egy „pillanatfelvételt" (*snapshot*) készít az aktuális kódról, amit vissza tudsz hívni a jövőben.
- **`git commit -a`**: Ez a parancs egy lépésben hozzáadja és commitolja a módosított fájlokat, kihagyva a külön `git add` lépést.

3. Local repository (helyi tárhely)

A helyi tárhely (vagyis a projekted mappája) tartalmazza az összes eddigi commitodat, amelyeket a saját gépeden tárolsz. Ezek a változások nem kerülnek automatikusan fel a távoli szerverre.

- **`git push`**: A helyi commitokat feltölti a **remote repository**-ba (távoli tárhelyre), ezzel például mások is elérhetik a változtatásokat.

4. Remote repository (távoli tárhely)

Ez az a Git repository, amely egy távoli szerveren található. Ez lehet például a GitHub repository-d (de megeshet, hogy GitLabon, BitBucketen stb. található). Más fejlesztők is elérhetik, és itt osztják meg egymással a kódot.

- **`git fetch`**: Ezzel a paranccsal letöltheted a távoli repository-ban történt változásokat, de nem frissíti automatikusan a helyi munkakönyvtáradat.
- **`git pull`**: Ez a parancs egyesíti a távoli változtatásokat a helyi repository-val és workspace-szel is. Alapvetően a `git fetch` és a `git merge` egy lépésben történő végrehajtása.
- **`git clone`**: Egy távoli repository teljes másolatát hozza létre a helyi gépen.

5. Reset és Diff parancsok

- **`git reset <fájl>`**: Visszavonja a staging területre történő fájlhozzáadást, azaz eltávolítja a fájlt az előkészítő területről.
- **`git reset <commit>`**: Visszaállítja a helyi repository állapotát egy korábbi commitra, a változtatások attól függően lehetnek visszavonva.
- **`git diff`**: Megmutatja a workspace és a staging terület közötti különbségeket.
- **`git diff HEAD`**: Megmutatja a különbségeket a workspace és a legutóbbi commit (HEAD) között.

6. Összefoglalás

- A fejlesztő, vagyis te a **workspace**-ben dolgozol.
- A fájlokat a **staging** területre kell „küldeni”, mielőtt commitolnád őket.
- A commitok a **local repository**-ban tárolódnak.
- A **remote repository** egy távoli szerveren található, és a `git push` parancs segítségével lehet a helyi commitokkal frissíteni a távoli repositoryt.

Most hogy tisztáztuk a git működését, nézzük meg, hogyan lehet telepíteni és használni néhány egyéb parancsot is.

Telepítés, konfigurálás és parancsok

1. Git telepítése és konfigurálása

Ha még nincs telepítve a Git, először le kell töltened és telepítened kell:

- **Linux**:

Használd a disztribúciód által szolgáltatott package managert, például:

```
sudo apt-get install git
sudo pacman -Sy git
sudo dnf install git
```

- **MacOS**:

Használd a Homebrew-t:

```
brew install git
```

- **Windows**:

Töltsd le a Git for Windows programot és telepítsd. Vagy ha telepítve van a Chocolately package manager, akkor a következő parancs is feltelepíti:

```
choco install -y git
```

Miután telepítetted a Gitet, be kell állítanod a felhasználóneved és az email címed.

```
git config --global user.name "Név"
git config --global user.email "email@example.com"
```

Ezek az adatok minden egyes commitban megjelennek majd.

GitHub Desktoppal

Lehet telepíteni és használni a Gitet GUI segítségével is, ehhez a GitHub Desktop nevű programot szükséges letölteni innen. Ez elvégzi nekünk a konfigurációt is, amikor legelőször belépünk, ezért nem kell a fenti konfigurációs parancsokat lefuttatni.

2. Git repository létrehozása

Új repository létrehozása

Ha egy új projektet kezdesz, futtasd ezt a parancsot a projekt mappájában:

```
git init
```

Ez létrehoz egy új Git repository-t az aktuális mappában.

Létező repository klónozása

Ha már létezik egy repository, és le szeretnéd másolni, használd a következőt:

```
git clone <repository_url>
```

Például:

```
git clone https://github.com/felhasznalo/repository
```

3. Távoli repository (remote) kezelése

Távoli repository hozzáadása

Ezzel a paranccsal hozzáadhatsz egy távoli repository-t:

```
git remote add origin <url>
```

Változások feltöltése a távoli repository-ba

Miután commitoltad a változásokat, feltöltheted őket a távoli szerverre:

```
git push origin <ág_név>
```

Változások letöltése a távoli repository-ból

A legújabb változásokat a távoli repository-ból letöltheted ezzel:

```
git pull origin <ág_név>
```

4. Fájlok kezelése

Fájlok hozzáadása a staging areához

Mielőtt commitolnád a változásokat, hozzá kell adnod őket a staging areához:

```
git add <fájlnév>
```

Az összes fájl hozzáadásához használd:

```
git add .
```

Változások commitolása

Miután hozzáadtad a fájlokat a staging areához, commitolhatod őket.

```
git commit -m "Commit üzenet"
```

Módosítások megtekintése

A változásokat a `status` paranccsal ellenőrizheted:

```
git status
```

A módosított fájlok tartalmát pedig ezzel a paranccsal láthatod:

```
git diff
```

5. Ágak kezelése

Új ág létrehozása

Az ágak lehetővé teszik, hogy különböző funkciókon dolgozz anélkül, hogy befolyásolnád a fő kódot.

```
git branch <ág_név>
```

Átváltás másik ágra

Ha egy másik ágon szeretnél dolgozni:

```
git checkout <ág_név>
```

Új ág létrehozása és átváltás egy lépésben

```
git checkout -b <ág_név>
```

Ágak egyesítése

Miután befejezted a munkát egy ágon, összeolvashatod a módosításokat a fő ággal (általában a `main` vagy `master` ággal):

```
git merge <ág_név>
```

6. Változások visszaállítása

Legutóbbi commit visszaállítása

Ha hibát követtél el, visszavonhatod a legutóbbi commitot, miközben megtartod a fájlokat a staging areában:

```
git reset --soft HEAD~1
```

Ha teljesen vissza akarod állítani a commitot és a módosításokat:

```
git reset --hard HEAD~1
```

Staging area kiürítése

Ha véletlenül rossz fájlokat adtál a staging area-hoz:

```
git reset <fájlnév>
```

Fájlok visszaállítása a legutóbbi commit szerinti állapotba

Ha például módosítottál egy fájlt, de nem szeretnéd azt megtartani, vagy éppenséggel kitöröltél egy fájlt és mégis szükség van rá, akkor ez a parancs visszaállítja azt neked. Ez csak abban az esetben működik, ha nincsenek az új módosítások staging areában.

```
git checkout -- <fájlnév>
```

7. Git log és történet megtekintése

Commitok listázása

A `log` paranccsal megtekintheted a commitok listáját:

```
git log
```

Ha rövidebb formában szeretnéd látni:

```
git log --oneline
```

Ágak összehasonlítása

Ha szeretnéd összehasonlítani, mi változott két ág között:

```
git diff <ág1> <ág2>
```

8. Git stash – Változások ideiglenes mentése

Ha változásaid vannak, de még nem akarod commitolni őket, használhatod a `stash` parancsot, hogy ideiglenesen elmentsd őket:

```
git stash
```

Ha vissza akarod állítani a stashed változásokat:

```
git stash pop
```

`git stash apply` is használható. A kettő között az a különbség, hogy az `apply` nem törli ki a stashból a változtatásokat, míg a `pop` ki is törli őket.

9. Segítség kérése

Ha elakadnál, a Git beépített súgóját is használhatod:

```
git help <parancs>
```

Például, ha többet szeretnél tudni a `commit` parancsról:

```
git help commit
```

10. Egyes commitok áthelyezése másik branchre

A `git cherry-pick` parancs lehetővé teszi, hogy egy (vagy több) commitot egy másik branch-be átemeljünk. Akkor használjuk, ha egy adott commitot szeretnénk átvinni egy másik branchre anélkül, hogy a teljes történetet átvinnénk.

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

Például ha egy fejlesztés az `feature-branch` branch-en van, de egy adott commitot át szeretnénk vinni a `main` branchre (ha többet szeretnénk átvinni, elég csak felsorolni őket):

```
git checkout main
```

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

11. Branch áthelyezése

A `git rebase` arra szolgál, hogy a commitokat egy másik branch végére igazítsa. Általában kétféle módon használjuk: `interactive rebase`, illetve az egyszerű `rebase`.

Egyszerű rebase

```
git checkout <branch>
```

```
git rebase <base-branch>
```

Például ha a `feature-branch` branch-en dolgozunk, és szeretnénk a legfrissebb `main` branch változásokat átemelni:

```
git checkout feature-branch
```

```
git rebase main
```

Ez a parancs a `feature-branch` commitjait a `main` branch aktuális állapota mögé igazítja, így elkerülve az extra merge commitokat.

Interaktív rebase

Az interaktív rebase lehetőséget ad a commitok újrarendezésére, szerkesztésére, törlésére vagy kombinálására.

```
git rebase -i <commit-hash>
```

Például a következő parancs az utolsó 5 commitot listázza és lehetőséget biztosít azok szerkesztésére:

```
git rebase -i HEAD~5
```

A megnyíló szövegszerkesztőben többféle műveletet végezhetünk:

- **pick**: megtartjuk a commitot.
- **reword**: szerkesztjük a commit üzenetét.
- **edit**: megváltoztatjuk a commit tartalmát.
- **squash**: összevonunk két commitot.
- **drop**: töröljük a commitot.

Ha több commitot szeretnénk átvinni egy másik branchre, először a `git cherry-pick` használható, majd szükség esetén a commitok rendezésére vagy finomítására `git rebase`-t alkalmazhatunk.