Uživatelská konfigurace

▶ Porovnání souborů přes aplikaci Meld

Úložiště

Vytvořit na lokálním prostředí

```
git init --bare <cesta>
```

> **Warning**
>
> `<cesta>` = vytvoří úložiště do cesty, musí mít na konci cesty `.git`

Použít do pracovního prostředí

```
git clone <cesta>
```

> **Warning**
>
> `<cesta>` = adresa k úložišti, musí mít na konci cesty `.git`

> **Tip**
>
> Cesta může být lokální i online.

Git Submodules

Umožňuje vložit jeden Git repozitář do jiného jako podadresář, přičemž si oba repozitáře zachovávají nezávislost.

Submoduly řeší problém, kdy potřebujete:

- Zahrnout externí kód do svého projektu
- Udržovat přesnou verzi závislostí
- Pracovat na více souvisejících projektech současně

▶ Základní struktura

▶ Základní příkazy

▶ Praktický příklad použití v Unity projektu

▶ Tipy pro práci se submoduly

▶ Časté problémy a řešení

▶ Výhody a nevýhody

Git Flow

Git Flow je strategie pro správu větví v Gitu, která usnadňuje práci v týmech a řízení verzí softwaru.

▶ Základní větve v Git Flow

▶ Pomocné větve

▶ Vývoj nové funkce

▶ Příprava vydání

▶ Oprava chyby v produkci

▶ Pravidla pro práci s Git Flow

▶ Vizualizace Git Flow

Pro více informací navštivte oficiální dokumentaci Git Flow

Pro umístění (pushnutí) nové větve `develop` na Git server (např. GitHub, GitLab) postupuj takto:

1. Vytvoř branch `develop` (pokud ji ještě nemáš):

   ```
   git checkout -b develop
   ```

2. Ujisti se, že máš vzdálený repozitář:

   ```
   git remote -v
   ```

   > **Note**
   >
   > Pokud není nastaven, použij např. `git remote add origin <url>`

3. Pushni větev `develop` na remote:

   ```
   git push -u origin develop
   ```

   > **Note**
   >
   > Parametr `-u` nastaví `develop` jako výchozí sledovanou větev vůči `origin/develop`.

4. Ověř, že větev je online:

   Lze se nyní podívat na webové rozhraní (GitHub, GitLab apod.), jestli se větev vzdáleně objevila.

> **Warning**
>
> Smazání vzdálené větve je nevratná operace.
>
> Ujisti se, že větev už nepotřebuješ a že všechny potřebné změny byly začleněny do jiných větví.

Pro smazání vzdálené větve z Git serveru (např. GitHub, GitLab) postupuj takto:

1. Zobraz si seznam všech větví (lokální i vzdálené):

```
git branch -a
```

2. Smaž vzdálenou větev jedním z těchto příkazů:

```
git push origin --delete <nazev-vetve>
# nebo kratší varianta
git push origin :<nazev-vetve>
```

> **Note**
>
> Nahraď `<nazev-vetve>` skutečným názvem větve, kterou chceš smazat.

3. Vyčisti lokální reference na smazané vzdálené větve:

```
git fetch --prune
```

> **Tip**
>
> Tento krok není povinný, ale pomáhá udržet lokální repozitář čistý.

🟩 Kdy použít Pull Request (PR):

- **Týmová spolupráce** – ostatní vývojáři mají možnost kód zkontrolovat.
- **Code review**, komentáře, schválení.
- **Automatizované testy a CI/CD pipeline** se spustí před sloučením.
- **Audit a historie** – vše je dohledatelné, kdo co schválil a proč.

> ✅ **Typické v profesionálních projektech**, GitHub, GitLab, Bitbucket atd.

🟨 Kdy použít přímo `git merge`:

- **Jsi sám na projektu** nebo děláš rychlý merge bez potřeby review.
- Chceš rychle sloučit větev lokálně (např. `feature/login` do `develop`).
- Nepotřebuješ historii PR ani schvalovací proces.

```
git checkout develop
git merge feature/login-page
```

> ⚠️ Tento merge probíhá bez schválení a obvykle **lokálně** – pak je třeba pushnout změny.

🧠 Shrnutí

Scénář	Použij PR?	Použij Merge?
Pracuješ sám	❌	✅
Týmový projekt	✅	❌ *(většinou)*
Chceš review, testy, audit	✅	❌
Lokální rychlé sloučení	❌	✅

Aktualizace .gitignore

Odstraňte mezipaměť všech souborů:

```
git rm -r --cached .
```

Jakmile vymažete existující mezipaměť, přidejte/stage soubor/soubory v aktuálním adresáři:

```
git add .
```

Potvrďte změny:

```
git commit -m "Aktualizace .gitignore"
```

Nová Branch

```
git checkout master       # Přepne se do zdrojové branch
git branch newbranch      # Vytvoří branch a tím se uloží stejné commity z předchozí
aktivní branch
git checkout master       # Přepne se do zdrojové branch
git reset --hard HEAD~3   # Odstraní 3 commity zpět.
git checkout newbranch    # Přepne se do cílové branch
```

Více info zde.

Existující Branch

```
git checkout existingbranch # Přepne se do cílové existující branch
git merge branchToMoveCommitFrom # Přesune commity ze zdrojové branch
git checkout branchToMoveCommitFrom # Přepne se do zdrojové branch
git reset --hard HEAD~3 # Odstraní 3 commity zpět
git checkout existingbranch # Přepne se do cílové existující branch
```

Více info zde.

`fixup!`

= Nepoužije zprávu z commitu do opravy

```
git commit --fixup <hashId>
```

nebo

```
git commit -m "fixup! <hashId> notUsedMessage"
```

`squash!`

= Sloučí zprávu z commitu do opravy

```
git commit -m "squash! <hashId> optionalCustomMessage"
```

▶ Příklad

Nahrazení Vzdálené Branch z Lokální Branch

1. Přepnout se na novou branch

```
git checkout --orphan latest_branch
```

> **Note**
>
> `--orphan` znamená, že vytvoří branch bez historie commitů

2. Přidat všechny soubory.

```
git add -A
```

3. Provedení commitu.

```
git commit -am "Initialize commit"
```

> **Tip**
>
> `-am` je zkrácený zápis
>
> Je to stejné jako zápis: `--all` `--message "commit message"`

4. Smazat hlavní branch.

> **Warning**
>
> Zjistěte název hlavní větve. (Většinou se jmenuje `master` nebo `main`)

```
git branch -D main
```

5. Přejmenovat aktivní branch na branch z předchozího kroku.

> **Warning**
>
> Zjistěte název hlavní větve. (Většinou se jmenuje `master` nebo `main`)

```
git branch -m main
```

6. Odeslat změny z pracovního adresáře do centrálního úložiště

```
git push -f origin main
```

> **Tip**
>
> `-f` (force) = Historie commitů v centrálním úložišti je nahrazena historií z pracovního adresáře