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.

Co jsou Git Submodules

Submoduly řeší problém, kdy potřebujete:

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

Základní struktura

```
HlavniProjekt/
  ├── .git/                  # Git repozitář hlavního projektu
  ├── .gitmodules            # Konfigurace submodulů
  ├── BeznyAdresar/          # Běžný adresář v hlavním projektu
  └── Submodul/              # Podadresář obsahující submodul
      └── .git/              # Samostatný git repozitář
```

Základní příkazy

Přidání submodulu

```
# Syntaxe: git submodule add [URL] [cesta]
git submodule add https://github.com/uzivatel/knihovna libs/knihovna
```

Klonování projektu se submoduly

```
# Způsob 1: Vše najednou
git clone --recursive https://github.com/uzivatel/projekt

# Způsob 2: Nejprve projekt, pak inicializace submodulů
git clone https://github.com/uzivatel/projekt
git submodule init
git submodule update
```

Aktualizace submodulů

```
# Aktualizace všech submodulů na nejnovější commit z remote
git submodule update --remote

# Aktualizace konkrétního submodulu
cd cesta/k/submodulu
git checkout main
git pull
cd ../..
git add cesta/k/submodulu
git commit -m "Aktualizován submodul na nejnovější verzi"
```

Praktický příklad použití v Unity projektu

Vhodná struktura

```
UnityProjekt/
  ├── .git/
  ├── .gitmodules
  └── Assets/
      ├── Game/             # Váš herní kód (součást hlavního repozitáře)
      └── Plugins/          # Složka pro externí knihovny
          ├── UI-Framework/ # Submodul s UI frameworkem
          │   └── .git/
          └── Network/      # Submodul s síťovou knihovnou
              └── .git/
```

Vytvoření této struktury

```
# Vytvoření hlavního repozitáře
cd UnityProjekt
git init

# Přidání UI frameworku jako submodulu
git submodule add https://github.com/author/ui-framework Assets/Plugins/UI-Framework

# Přidání síťové knihovny jako submodulu
git submodule add https://github.com/author/network-lib Assets/Plugins/Network
```

Tipy pro práci se submoduly

1. **Přepínání mezi verzemi**

```
cd cesta/k/submodulu
git checkout v2.0.0
cd ../..
git add cesta/k/submodulu
git commit -m "Změněna verze submodulu na v2.0.0"
```

2. **Úpravy v submodulu**

```
# Změny v submodulu
cd cesta/k/submodulu
git checkout -b oprava-chyby
# proveďte změny
git commit -am "Oprava chyby"
git push origin oprava-chyby
# vytvořte pull request v repozitáři submodulu
```

3. **Odstranění submodulu**

```
# 1. Odebrat z .gitmodules
git submodule deinit cesta/k/submodulu
# 2. Odebrat z .git/config
git rm --cached cesta/k/submodulu
# 3. Smazat adresář
rm -rf cesta/k/submodulu
rm -rf .git/modules/cesta/k/submodulu
# 4. Commit
git commit -m "Odstraněn submodul"
```

Časté problémy a řešení

1. **Submodul v "detached HEAD" stavu**

```
cd cesta/k/submodulu
git checkout main
```

2. **Submodul ukazuje změny i když žádné nemáte**

```
git submodule update
```

3. **Změna URL submodulu**

```
git config --file=.gitmodules submodule.nazev.url NOVA_URL
git submodule sync
```

4. **Kontrola stavu submodulů**

```
git submodule status
```

Výhody submodulů

- Přesná kontrola verzí externích knihoven
- Možnost přímo upravovat a přispívat do závislostí
- Lepší organizace kódu v komplexních projektech

Nevýhody submodulů

- Vyšší složitost správy repozitáře
- Nutnost vždy aktualizovat hlavní repozitář po změnách v submodulech
- Strmější učící křivka pro nové členy týmu

Git Flow - Jak správně pracovat s větvemi

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

Hlavní větve

- `main` **(nebo** `master`**)**: Obsahuje produkční verzi kódu
- `develop`: Obsahuje připravované změny pro příští vydání

Pomocné větve

- `feature/*`: Pro vývoj nových funkcí
- `release/*`: Pro přípravu vydání
- `hotfix/*`: Pro rychlé opravy chyb v produkci

Jak pracovat s Git Flow

1. Vývoj nové funkce

```
# Vytvoření nové feature větve
git checkout develop
git checkout -b feature/nova-funkce

# Po dokončení vývoje
git checkout develop
git merge feature/nova-funkce
```

2. Příprava vydání

```
# Vytvoření release větve
git checkout develop
git checkout -b release/1.0.0

# Po dokončení příprav
git checkout main
git merge release/1.0.0
git checkout develop
git merge release/1.0.0
```

3. Oprava chyby v produkci

```
# Vytvoření hotfix větve
git checkout main
git checkout -b hotfix/oprava-chyby

# Po dokončení opravy
git checkout main
git merge hotfix/oprava-chyby
git checkout develop
git merge hotfix/oprava-chyby
```

Pravidla pro práci

1. Nikdy nepracujte přímo v `main` ani `develop` větvích
2. Každá funkce má vlastní feature větev
3. Před sloučením proveďte code review (to znamená, že někdo zkontroluje váš kód a schválí ho před sloučením)
4. Po sloučení release nebo hotfix větve označte verzi pomocí Git tagu

---

Pro více informací navštivte oficiální dokumentaci Git Flow

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