Strukturování a psaní commit zpráv

- **Jednoznačnost:** Každý commit by měl popisovat jednu konkrétní změnu. Rozdělte větší úpravy do menších kroků.
- **Začátek akcí:** Začínejte zprávu slovem, které označuje typ změny (např. "ADD", "FIX", "UPD").
- **Stručnost:** Popis by měl být krátký a informativní (doporučuje se do 50 znaků v první větě).
- **Dodatečný kontext:** Pokud je potřeba, přidejte podrobný popis v další části commit zprávy (oddělte jedním prázdným řádkem).

Příklad správné struktury:

```
ADD: Implementace nového API pro správu uživatelů

Přidáno API pro vytváření, mazání a aktualizaci uživatelů.
Zahrnuje testy a dokumentaci.
```

1. **Klonování projektu:** Načtěte projekt lokálně pomocí:

   ```
   git clone <URL repozitáře>
   ```

2. **Pravidelně aktualizujte svou lokální větev:** Před začátkem práce se ujistěte, že máte aktuální kód:

   ```
   git checkout main
   git pull origin main
   ```

3. **Pracujte na vlastní větvi:** Každý úkon (funkce, oprava chyby, změna) by měl být implementován na samostatné větvi.

   ```
   git checkout -b <typ-změny>/<krátký-popis>
   ```

   Například:

   ```
   git checkout -b feature/pridani-loginu
   ```

4. **Commitujte pravidelně:** Každý commit by měl obsahovat jednu logickou změnu a měl by mít jasný popis:

   ```
   git add .
   git commit -m "ADD: Implementace přihlášení uživatele"
   ```

5. **Pushnutí změn do vzdáleného repozitáře:** Po dokončení změn odešlete svou větev na GitHub:

   ```
   git push origin <nazev-vetve>
   ```

2. Pravidla pro názvy větví a commitů

Názvy větví: Používejte popisné názvy s předponou podle typu změny:

- `feature/` – Pro nové funkce. *Např.:* `feature/pridani-loginu`
- `bugfix/` – Pro opravy chyb. *Např.:* `bugfix/oprava-chyby-prihlaseni`

- `update/` – Pro úpravy existujících funkcí. *Např.:* `update/zmena-nacitani-konfigurace`

Popisy commitů: Začínejte akcí (anglicky nebo česky) a stručně popište změnu:

- `ADD:` Pro přidání nových funkcí.
- `FIX:` Pro opravy chyb.
- `UPD:` Pro úpravy funkcí.
- `REFACTOR:` Pro refaktorování kódu.
- `REMOVE:` Pro odstranění nepotřebného kódu.

Příklad:

```
FIX: Opraveno špatné načítání konfigurace z config.yml
ADD: Přidána podpora pro více konfigurací
UPD: Změněno načítání světa na hodnotu z config.yml
REFACTOR: Restrukturalizace kódu pro čtení konfigurace
REMOVE: Odstraněny zastaralé metody načítání dat
```

3. Sloučení změn do hlavní větve

1. **Aktualizujte svou větev:**

   ```
   git checkout <nazev-vetve>
   git pull origin main
   git rebase main
   ```

2. **Přepněte se na hlavní větev a sloučte změny:**

   ```
   git checkout main
   git merge <nazev-vetve>
   ```

3. **Otestujte hlavní větev:** Po sloučení změn proveďte testování na hlavní větvi, abyste zajistili, že všechny nové změny fungují správně a nedošlo k nečekaným problémům.

4. **Odešlete změny do vzdáleného repozitáře:**

   ```
   git push origin main
   ```

5. **Aktualizujte svou větev:**

   ```
   git checkout <nazev-vetve>
   git pull origin main
   git rebase main
   ```

6. **Přepněte se na hlavní větev a sloučte změny:**

   ```
   git checkout main
   git merge <nazev-vetve>
   ```

7. **Odešlete změny do vzdáleného repozitáře:**

```
git push origin main
```

5. Řešení konfliktů

Pokud dojde ke konfliktům:

1. Git označí konfliktní soubory. Použijte příkaz `git status`, abyste viděli, které soubory jsou konfliktní:

   ```
   git status
   ```

2. Opravte konflikty manuálně v označených souborech (hledání `<<<<<<<`, `=======`, `>>>>>>>`).

3. Po úpravě ověřte, že jsou všechny konflikty vyřešeny, pomocí příkazu `git status`. Ujistěte se, že žádné konfliktní soubory nezůstaly.

4. Přidejte upravené soubory:

   ```
   git add <soubor>
   ```

5. Commitněte změny:

   ```
   git commit -m "Vyřešení konfliktů pro sloučení s main"
   ```

Pokud dojde ke konfliktům:

1. Git označí konfliktní soubory. Opravte je manuálně.

2. Po úpravě commitněte změny:

   ```
   git add .
   git commit -m "Vyřešení konfliktů pro sloučení s main"
   ```