

GIT – krok za krokem

Lukáš Růžička

prosinec 2020

Co je GIT?

- verzovací systém
- open source
- s aktivním vývojem
- vytvořil je Linus Torvalds
- distribuovaný (necentralizovaný)

Proč?

Vytváření věcí duševní povahy často zahrnuje:

- přepisování
- mazání
- zkoušení slepých cest
- návraty k předchozímu
- revize a korekce od spolupracovníků

Srovnání s kdysi

pravěk

opište si zadání
průběžně odevzdávejte
zkus ještě další verzi
použij obě verze, přepiš
tady jsou opravy, přepiš
raději bych přece jen tu dřívější verzi
zahod' to
cos to zahodil, ty jelito?

gitvěk

fork
git commit
git branch
git merge
git merge
git reset
git rm
git clone

Prostě ... **Vždycky GIT!**

Co potřebujeme?

- účet na nějakém serveru s gitem (Pagure, Github, Bitbucket, Atlasian)
- nainstalovaný git
- dobrý je i nějaký grafický Git editor (tig, gitk, gitg)

Instalace Gitu

- Fedora: `sudo dnf install git`
- Debian: `sudo apt-get install git`
- Mac: <http://git-scm.com/download/mac>
- Windows: <http://git-scm.com/download/win>
- github.com – založit účet

Konfigurace gitu

- `git config --global user.name "Your Name"`
- `git config --global user.email "yourname@example.com"`

Vytváříme první repozitář

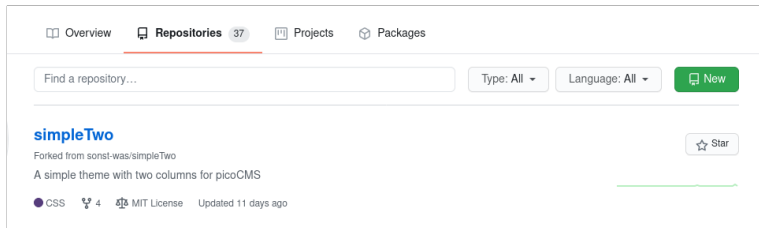
- 1 Běžte na stránku svého profilu.
- 2 Klikněte na **Repositories**.
- 3 Klikněte na **New**.
- 4 Vyplňte údaje.
- 5 Klikněte na **Create repository**.

Stránka profilu

The screenshot shows a web browser window displaying the GitHub profile of Lukáš Růžička. The browser's address bar shows the URL `https://github.com/lruzicka`. The GitHub navigation bar at the top includes links for Pull requests, Issues, Marketplace, and Explore. The profile section on the left features a circular profile picture with a red background and a white cross, the name **Lukáš Růžička**, the username `lruzicka`, an "Edit profile" button, and statistics: 2 followers, 0 following, and 1 star. Below this is a repository named "Red Hat". The main content area is titled "Overview" and lists "Popular repositories". These include:

- sphinx_asciidoc**: AsciDoc Builder and Writer for Sphinx (Python, 10 stars, 5 forks)
- numerology**: Numerology calculator in Python (Python, 1 star, 3 forks)
- talks**: Slides for various Linux related talks (Czech / English) (TeX, 1 star, 2 forks)
- exif_stats**: Read exif informations and create statistics. (Python, 1 star)
- rsted**: Forked from annu/rsted. Online reStructuredText editor (JavaScript)
- vimmodules**: Vim script

Nový repozitář




Vytvořit nový repozitář

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository.](#)

Owner *


Repository name *

 lruzicka ▾


 /

Great repository names are short and memorable. Need inspiration? How about **potential-octo-umbrella**?

Description (optional)

☒  **Public**

Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**

You choose who can see and commit to this repository.

Initialize this repository with:

Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a list of templates. [Learn more.](#)

☐ **Choose a license**
A license tells others what they can and can't do with your code. [Learn more.](#)

Create repository

Klonovat repozitář

Kopírujeme repozitář k sobě na disk.

- 1 Přístup SSH klíčem

```
git clone git@github.com:lruzicka/hokuspokus.git
```

- 2 Přístup heslem

```
git clone
```

```
https://github.com/lruzicka/hokuspokus.git
```

Iniciace repozitáře

Pokud nemůžeme z nějakého důvodu repozitář naklonovat, nebo chceme vytvořit pouze lokální repozitář (bez serveru), můžeme repozitář tzv. **založit** (initiate) příkazem

- `git init`

Zobrazení konkrétní situace

Zjišťujeme situaci na lokální kopii. Příkaz nám ukáže základní informace o repozitáři

- aktuální větev
- stav větve (aktuální, zaostává, předchází se)
- stav rozdělané práce

Zobrazení historie

Zobrazíme záznam **historie** pro aktuální větev daného repozitáře od nejnovějšího po nejstarší.

- commit (hash)
- autor commitu
- datum commitu
- název commitu (commit message)
- `git log --oneline`

Prohlížení commitu

Zobrazíme obsah daného commitu.

- `git show HEAD~2`
- `git show 622a1fee15c347`

Úprava obsahu

Obsah v gitu může být jakýkoliv, takže jej upravujeme patřičným způsobem, jak potřebujeme a jak jsme zvyklí.

- textový editor
- textový procesor
- specifická aplikace

Git si každé změny všimne.

Untracked a Staging

Když provedeme nějaké úpravy, například přidáme soubor, git o něm neví a označuje ho jako **untracked**. Musíme o nich gitu říct.

Stejně tak úpravy provedené u známých souborů nejsou primárně zařazeny do zpracování, dokud to **explicitně neřekneme**.

Tento proces se nazývá **staging** . V analogii vlaku se jedná o *rezervaci jízdenek*.

- `git add novy-soubor`
- `git add adresar/*`
- `git add .`

Zapsání obsahu do stromu

Ačkoliv jsme Git informovali o změnách, dokud je nezapišeme do stromu historie, nejsou součástí vývojové větve.

V analogii vlaku je potřeba do vlaku *nastoupit*.

- `git commit`
- `git commit -m "Commit message"`

Nahrání na server

Lze samozřejmě Git používat pouze lokálně, ale chceme-li spolupracovat s dalšími lidmi, musíme naše změny nahrát na server.

- `git push`

To lze pouze tehdy, když mezi lokální a serverovou kopií nejsou žádné **konflikty**.

Synchronizace se serverem

Před každým zahájením práce je nutné stáhnout novinky ze serveru, aby se minimalizovalo riziko konfliktů.

- `git pull` (zkratka pro následující)
- `git fetch origin`
- `git rebase origin/branch` nebo
- `git merge origin/branch`

Použití jednotlivých strategií záleží na vkusu každého, jsou mezi nimi však určité rozdíly, které si ukážeme později.

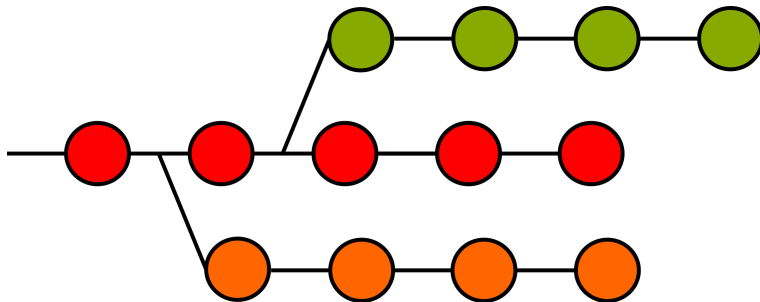
Práce s větvemi

Větve nám umožňují pracovat současně na projektu tak, abychom si navzájem nepřekáželi.

- Git podporuje neomezené množství větví
- větve umožňují oddělenou práci

Použití jednotlivých strategií záleží na vkusu každého, jsou mezi nimi však určité rozdíly, které si ukážeme později.

Příklad větvení



Zobrazení všech větví projektu

1 `git branch`

Změna aktuální větve

Do existující větve se lze přepnout příkazem **checkout**. Přepnutí do neexistující větve (vytvoření nové) vyžaduje navíc přepínač **-b**.

- 1 `git checkout branch`
- 2 `git checkout -b branch`

Nová větev se obsahově shoduje s aktuální větví až do místa rozvětvení. Lze tedy hned po vyvětvení pokračovat v práci.

Slučování větví

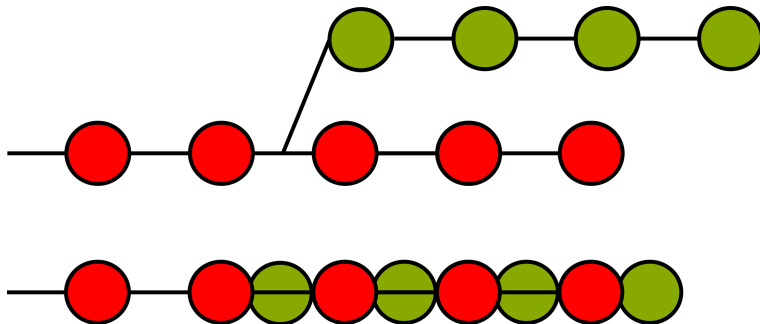
Větev můžeme sloučit s jinou větví a tak do jedné větve převzít obsah druhé větve, například do hlavní větve lze převzít vývoj nějakého doplňku z vedlejší větve.

Pro slučování větví existují dvě strategie:

- 1 merge (prorůstání)
- 2 rebase (roubování)

Slučování větví je možné pouze tehdy, když mezi větvemi **není konflikt**.

Příklad prorůstání



Slučování větví - prorůstání

Větve sloučíme následujícím postupem.

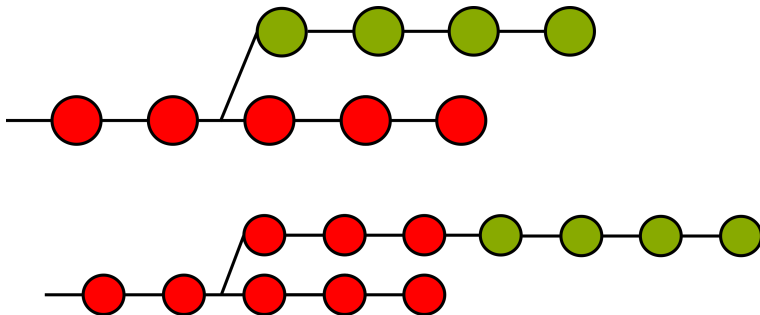
- 1 Přepneme se do cílové větve, to je ta, do které chceme slučovat

```
git checkout target-branch
```

- 2 Sloučíme obsah zdrojové větve do cílové větve

```
git merge source-branch
```

Příklad roubování



Slučování větví - roubování

Větve sloučíme následujícím postupem.

- 1 Přepneme se do zdrojové větve
`git checkout source-branch`
- 2 Naroubujeme zdrojovou větev na cílovou větev
`git rebase target-branch`
- 3 Přepneme se do cílové větve
`git checkout target-branch`
- 4 Sloučíme zdrojovou větev do cílové
`git merge source-branch`

Rozdíl mezi prorůstáním a roubováním

Prorůstání má následující vlastnosti, které roubování nemá:

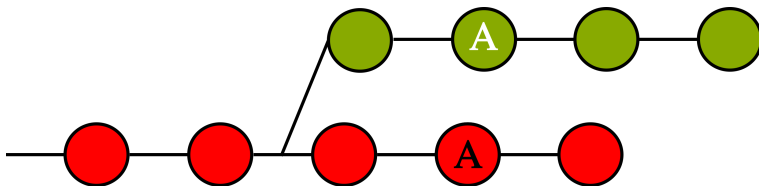
- zachování pořadí commitů
- **merge commit** v historii

Použití jednotlivých strategií závisí na osobních či projektových preferencích.

Konflikt slučování (merge conflict)

Jestliže se obsah dvou větví liší ve stejném místě, potom při slučování vznikne **konflikt**. V takovém případě se Git chová velmi konzervativně a **nedovolí sloučení**, dokud konflikt nebude vyřešen.

Důvod konfliktu



Řešení konfliktu

Kdykoliv Git hlásí konflikt při provádění slučovacího příkazu (merge, rebase), je potřeba jej nejprve ručně vyřešit, než můžeme větve sloučit.

- 1 Otevřeme soubor, ve kterém je konflikt.
- 2 Najdeme řádek <<<<<< HEAD
- 3 Najdeme řádek >>>>>>
- 4 A mezi nimi najdeme řádek =====
- 5 Všechno směrem nahoru je obsah původní větve.
- 6 Všechno směrem dolů je obsah nové větve.
- 7 Upravíme soubor tak, aby v něm zůstalo pouze to, co chceme. Uložíme soubor.

Řešení konfliktu

- 1 Řekneme gitu o změnách
`git add corrected-file`
- 2 Můžeme pokračovat ve slučování
`git merge --continue` nebo `rebase --continue`
- 3 Pokud se objeví další konflikt, vyřešíme jej a všechny další konflikty také.
- 4 Po vyřešení všech konfliktů obsahuje cílová větev změny ze zdrojové větve.

Smazání větve

Nepotřebnou větev můžeme smazat. Git nikdy nesmaže větev ze serveru, i když ji smažeme lokálně. Pro smazání ze serveru musíme použít jiný příkaz.

- Pro smazání sloučené větve. Git varuje, pokud větev není sloučená.

```
git branch -d branch-to-delete
```

- Pro smazání jakékoliv větve

```
git branch -D branch-to-delete
```

- Smazání větve na serveru

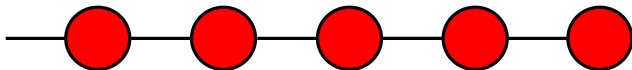
```
git push origin --delete branch-to-delete
```

Přidáváme k již uzavřenému commitu

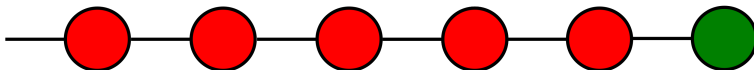
- `git commit --amend`
- upravíme commit message
- `git push` hlásí chybu ... proč?

Důvod konfliktu

Stav na serveru



Lokální stav



Vynucení změny

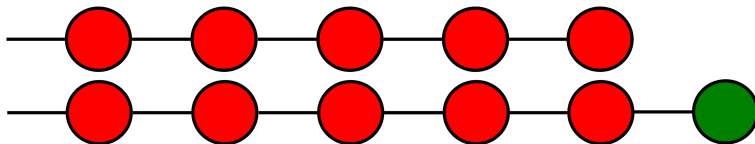
Pokud jsme si jisti, že naše lokální větev má správnou verzi a na serveru je verze nesprávná, můžeme Gitu přikázat, aby serverovou verzi přepsal verzí lokální. To může mít ale velké dopady na synchronizaci a proto to většinou děláme **pouze u soukromých větví**.

- `git push --force`

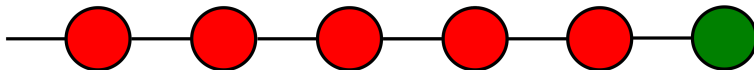
Vzdálená větev bude bez dalšího přepsána lokální variantou.

Konflikty po forcepush

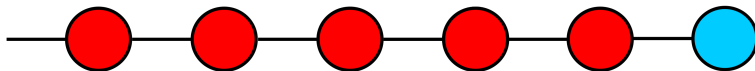
Stav na serveru



Spolupracovník 1



Spolupracovník 2



Měníme historii – interaktivní rebase

Při **interaktivním** rebase můžeme upravit historii commitů podle toho, jak potřebujeme:

- změnit commit message
- spojit několik commitů dohromady (squash)
- zahodit některé commity
- změnit strukturu repozitáře

Jakékoliv změny v historii větve **vyžadují forcepush** a mohou mít neblahé důsledky na všechny lidi, kteří na větvi pracují.

Git jako stroj času

Vzhledem k tomu, že si Git pamatuje všechny změny, které kdy ve větvi nastaly, tak je možné například

- porovnat dva body v čase nebo dvě větve
- vymazat přítomnost, vrátit se do historie, a začít znovu
- vrátit se do historie a vytvořit větev v místě návratu

Porovnání dvou verzí

Pomocí `git diff` si můžeme zobrazit rozdíly mezi dvěma prvky.

- `git diff hash1 hash2`
- `git diff origin/branch branch`
- `git diff branch1:file1 branch2:file1`

Smazání současnosti a návrat do minulosti

Můžeme si vybrat, na který commit se vrátíme a veškerý novější obsah zahodíme.

- `git reset 123456` (soft reset)
- `git reset --hard 123456` (hard reset)

Hard reset také navrátí stav souborového systému na stav dotyčeného commitu. **Soft reset** pouze posune ukazatel *HEAD*, ale obsah souborů zůstane nezměněn.

Pozor! `git reset` mění historii větve a porušuje její kontinualitu.

Smazání současnosti a návrat do minulosti

Pokud chceme smazat nějaký commit, ale zároveň nechceme, aby se změnila historie větve (například na *master* větvi), můžeme commit tzv. **revertnout**. To vytvoří nový commit, jehož obsah změní obsah větve na stav, jako by odstraňovaný commit ve větvi vůbec nebyl.

- `git revert 123456`

Je potřeba si uvědomit, že čím v historii vzdálenější commit chceme revertnout, tím více konfliktů budeme možná muset řešit. Někdy je tedy daleko lepší provést změny ručně a znovu je commitnout.

Návrat do minulosti a vyvětvění z toho místa

Někdy bychom se chtěli vrátit do minulosti a zkusit jinou cestu, ale zároveň si chceme uchovat, to, co už máme.

- Můžeme se tedy vrátit k určitému commitu

```
git checkout 123456
```

- A z toho místa vyvětvit novou větev

```
git checkout -b newbranch
```

Pak budeme mít jak původní větev se všemi změnami, tak původní větev s novými změnami.

Jak se vyhnout problémům – dobrá praxe

Problémům s konflikty se nelze vyhnout, ale lze je podstatně snížit, když dodržíme následující:

- pravidelně aktualizujeme master větev a rebasujeme na ni
- každý pracuje ve své vlastní větvi
- než commitnu změny, vždycky předtím pullnu
- nikdy neměníme obsah a historii master větve
- commitujeme menší celky, případný konflikt se řeší **hned**

Proč nejde pullnout nebo pushnout?

Někdy se, kvůli různým zásahům do historie repozitáře, odlišuje lokální od vzdálené větve. Potom Git **nedovolí** aktualizovat lokální větev pomocí `git pull`.

Mezi nejčastější důvody patří

- lokální větev je napřed před vzdálenou větví
- mezi lokální a vzdálenou větví je konflikt
- historie commitů se mezi větvemi liší

Někdy se Git pokusí situaci napravit sám, většinou je nutný zásah z venčí.

Lokální větev je novější než vzdálená

- `git push`

Lokální a vzdálená větev mají jinou historii

Toto se stane, když někdo provede nějaké změny historie na dotyčné větvi. Většinou by to neměl být problém, protože předpokládáme, že vzdálená větev je správná. V tom případě stačí naši větev resetnout podle vzdálené větve.

- `git fetch origin`
- `git reset --hard origin/branch`

Lokální a vzdálená větve mají jinou historii

Když mám nějaké lokální commity, které na vzdálené větvi nejsou a o které nechceme přijít. V tom případě

- Najdeme společný commit a naši větev softresetneme na tento commit.
`git reset 12fg56`
- Všechny naše úpravy odložíme do úschovny
`git stash`
- Potom resetneme na vzdálenou větev
`git reset --hard origin/branch`
- Aplikujeme změny z úschovny
`git stash apply`
- Commitneme a pushneme.

Vyzobnutí jednoho commitu do jiné větve.

Někdy můžeme mít v jedné větvi zajímavý commit, který chceme použít v jiné větvi, která se ale jinak zásadně liší. Takový commit si můžeme vyzobnout jako *třešničku na dortu*.

- `git checkout branch-to-keep`
- `git cherry-pick commit`
- Vyřešíme případně konflikty.

Otázky?

Kdo má otázky, ať se zeptá teď . . .

. . . nebo navždy pomlčí.

Děkuji za pozornost!