

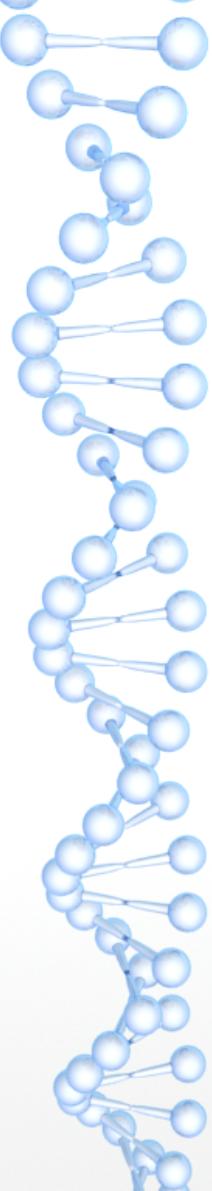


GIT

GIT primer

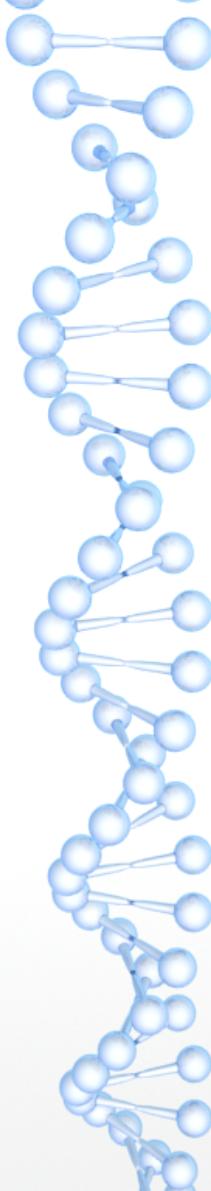
*Ako na GIT aj pre  
úplných začiatočníkov*

Miloš Korenčiak



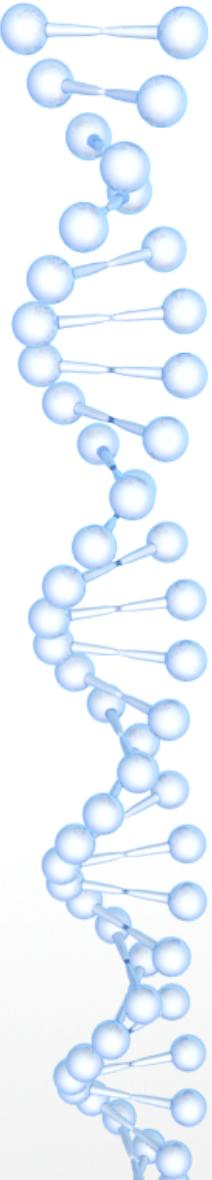
# Obsah

- 1. Čo (ne)urobí GIT za mňa?
- 2. Čo je vlastne GIT zač?
- 3. Tri stromy/databázy v GIT
- 4. Hands on lab (praktická ukážka toho najbežnejšieho na konzole)
- 5. Rôzne GUI pre GIT, aby sme to mali jednoduchšie
- 6. Sumár
- 7. Záver
- 8. Zdroje informácií



# 1. Čo (ne)urobí GIT za mňa?

- Neurobí za mňa prácu
- Nevyrieši komunikáciu v tíme ani nezhody v návrhu programu
- Nevyrieši každú kolíziu s kolegovým kódom
- Nevyrieši kóderove zabúdanie zálohovať /verzovať svoj program
- Manžérovi nepomôže zistiť, či sa projekt stíha

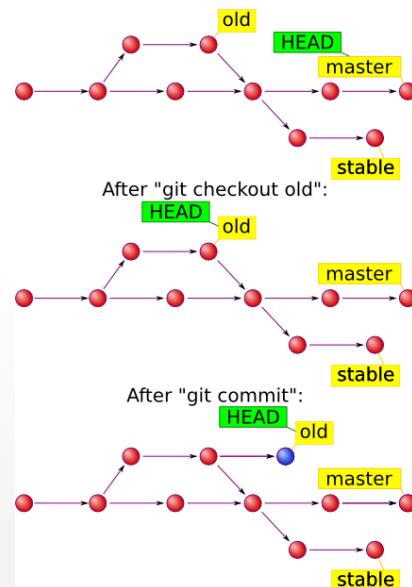


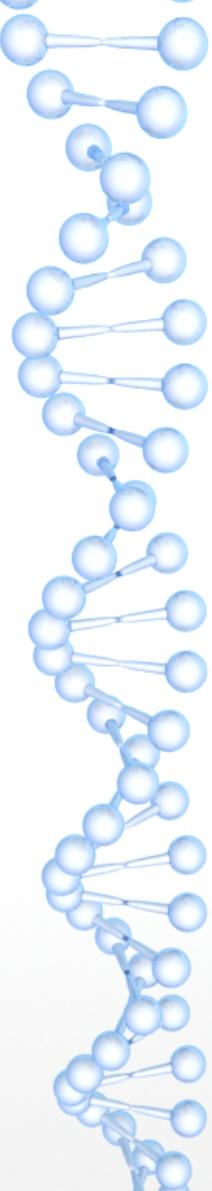
## 2. Čo je vlastne GIT zač?

- *a distributed revision control and source code management (SCM) system with an emphasis on speed*  
*(Wikipedia)*
- Umožňuje manažment kódu (SCM)
- Verzuje zdrojový kód
- Je distribuovaný
- Je rýchly :-) (za cenu...)
- Umožňuje mať rozrobených viac vecí naraz +prepínanie

# 2. Čo je vlastne GIT zač?

- Je to vlastne „strom“ commitov (check-point kódu)
- Priponá „stanice metra“
- Je uložený v internej databáze

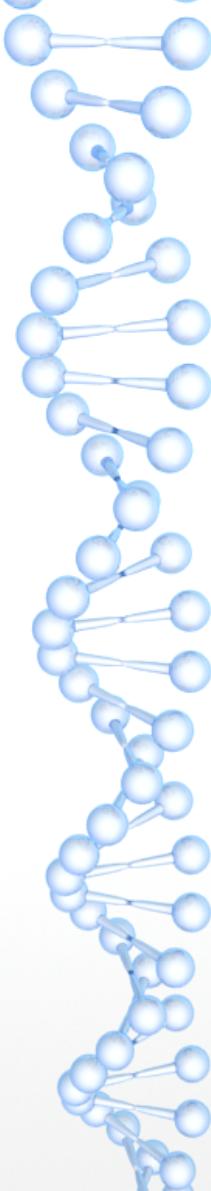




## 2. Čo je vlastne GIT zač?

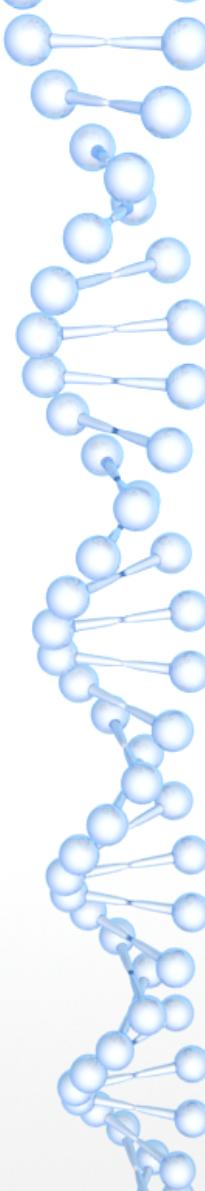
**Software tools for revision control are essential for the organization of multi-developer projects.**

(Wikipedia)



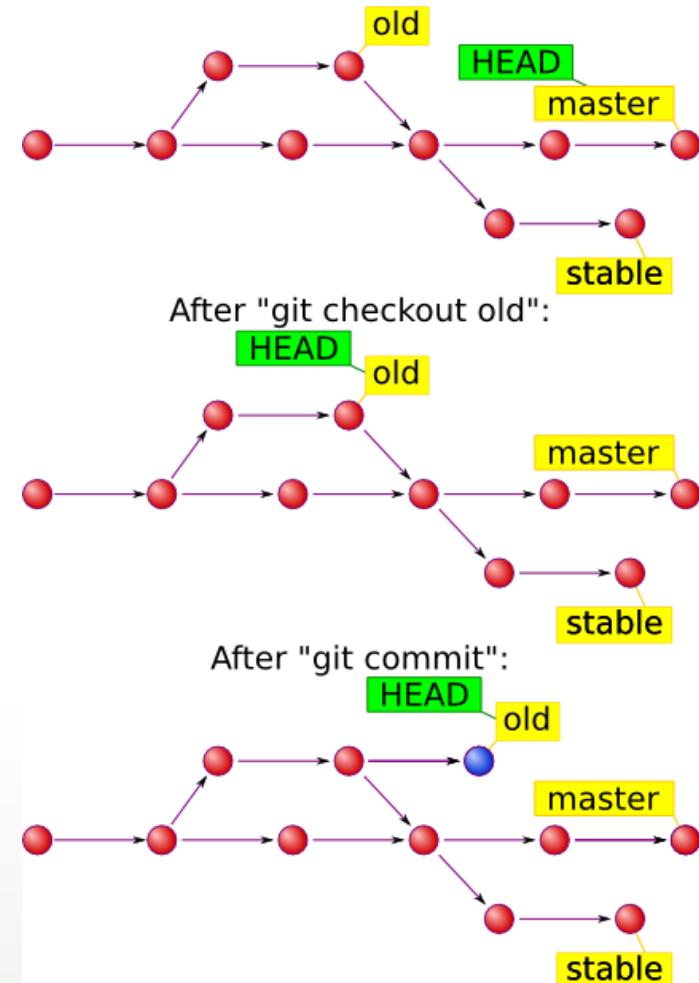
### 3. Tri stromy/databázy v GIT -working dir

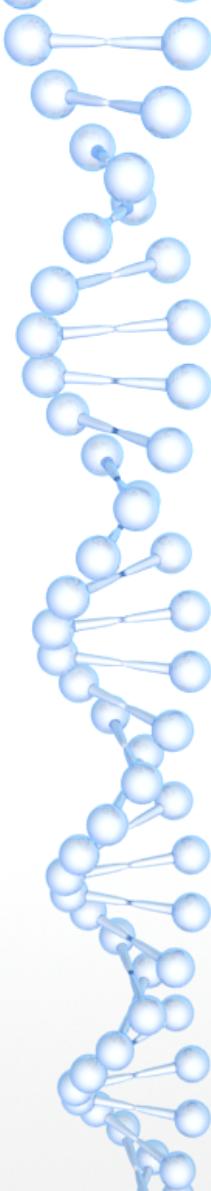
- Ten adresár v ktorom priamo programujeme
- Je v ňom teda fyzicky každý súbor projektu
- GIT prehľadáva aktuálny adresár a rekurzívne aj všetky podadresáre
- Zistuje, ktoré súbory oproti poslednému commitu pribudli (a potenciálne by sa mali pridať do repozitára), a ktoré súbory v repozitári (tzv. verzované /tracked súbory) sa zmenili
- GIT ho edituje pri checkout-e



### 3. Tri stromy/databázy v GIT -HEAD

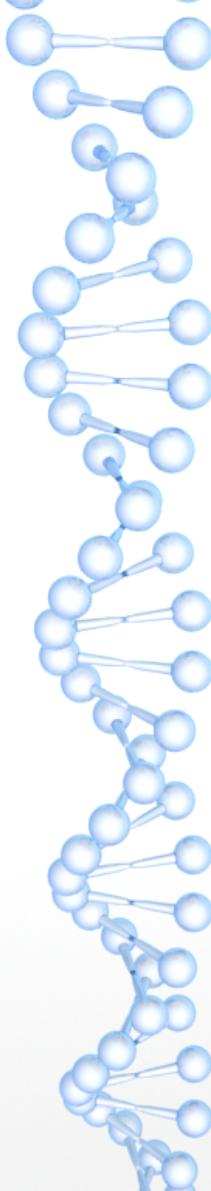
- Je to posledný commit v aktuálnej vetve „stromu“ (strom je celý GIT repozitár)
- Ak urobíme commit, stane sa synom aktuálneho HEAD commitu; HEAD ukazovateľ sa potom posunie na náš najnovší commit
- Ukazuje kde pracujeme





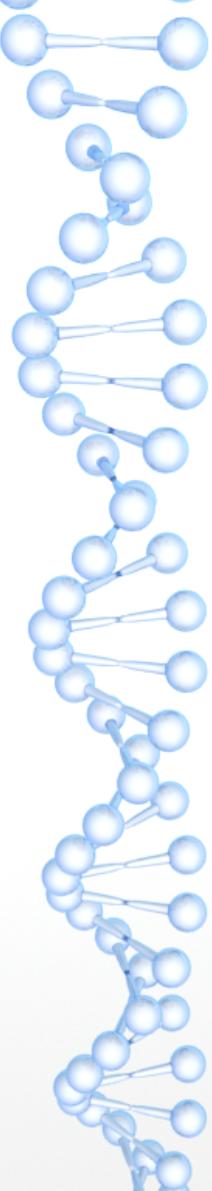
### 3. Tri stromy/databázy v GIT -index

- Je to miesto prípravy commitu (akýsi medzisklad)
- Pridávame tu zmeny, ktoré chceme zahrnúť do najbližšieho commitu
- Môžme tu pridať všetky zmeny zo súbora alebo len ich časť
- Môžeme pridať jeden alebo viac súborov
- Využívajú ho najmä GUI nástroje



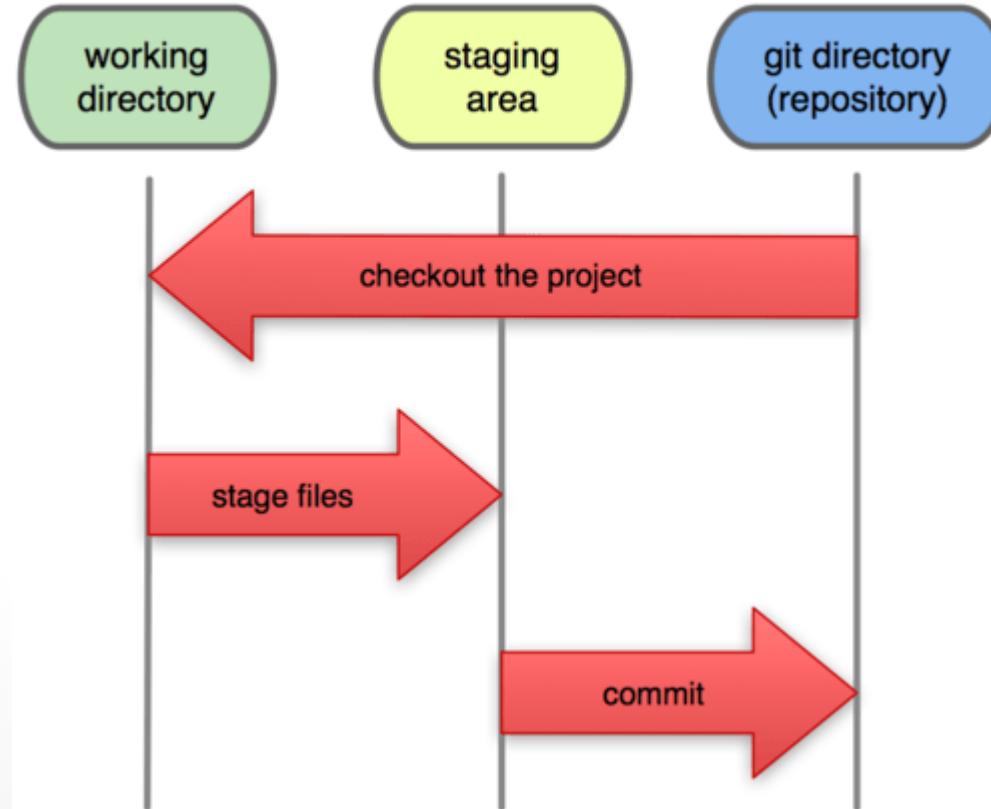
### 3. Tri stromy/databázy v Git -ich spolupráca

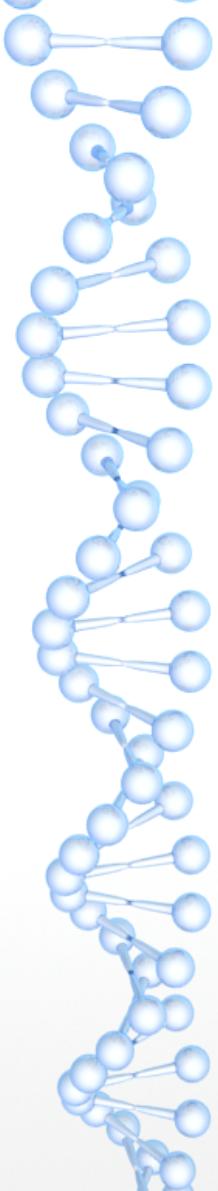
- Programovaním meníme working dir
- Príkazom git add (bude ukázaný neskôr) prenášame working dir zmeny do indexu
- Príkazom git commit (neskôr) prenášame index alebo aj working dir do stromu repozitára (až ten je verzovaný)
- IBA strom GIT repozitára je schopný priamo komunikovať so vzdialenými GIT repozitármi



### 3. Tri stromy/databázy v Git -ich spolupráca

#### Local Operations

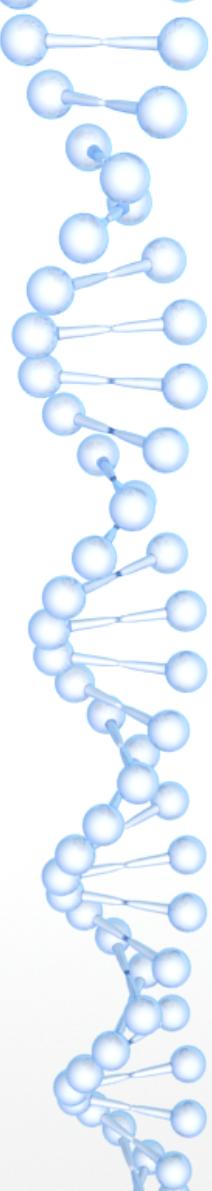




## 4. Hands on lab

(praktická ukážka v *git bash* konzole)

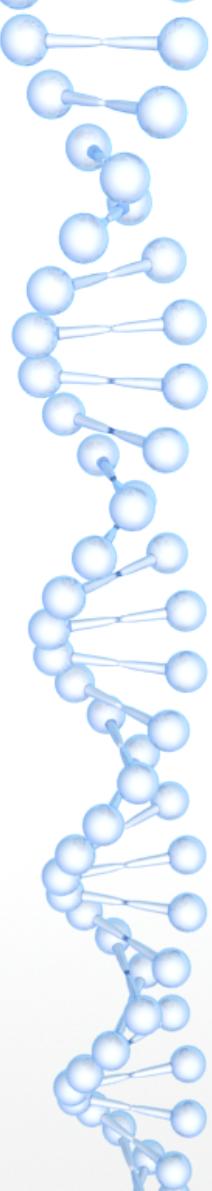
Predved'me si to na jednoduchom  
projekte ...



## 4.a) Vytvorenie repozitára

*\$ git init*

- Vytvorí v aktuálnom adresári lokálny git repozitár
- Jeho všetky dáta sú v jedinom *.git* skrytom podadresári

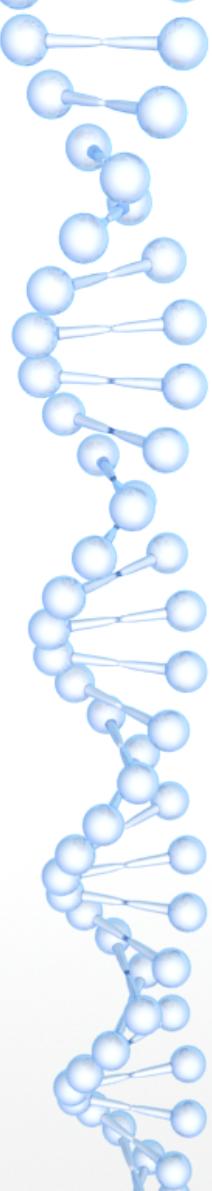


## 4.b) Stiahnutie repozitára

```
$ git clone https://github.com/requests/requests
```

```
$ git clone /home/m/kodenie/requests
```

- Naklonuje repozitár z daného URI
- Zvyčajne sa používa zabezpečený protokol
- Zvyčajne tento protokol vyžaduje autorizáciu pri posielaní zmien pri budúcom *push*



## 4.c) Commit +Add

`$ git status`

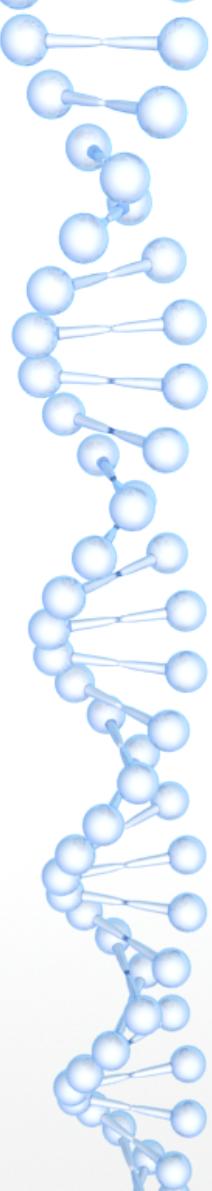
`$ git add haha.txt`

`$ git commit`

`$ git commit -a`

`$ git commit -a -m „ahoj -som popis commitu“`

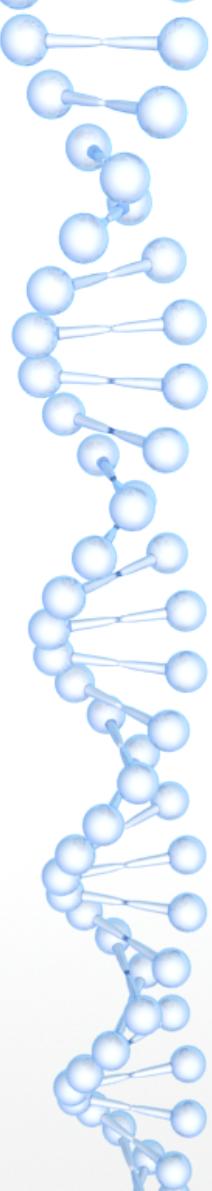
- Preskúmame stav repozitára
- Pridáme súbor, aby sa oň git staral
- Commit-neme jeho zmeny do git-u („Tri stromy git-u“)



## 4.d) Push

*\$ git push*

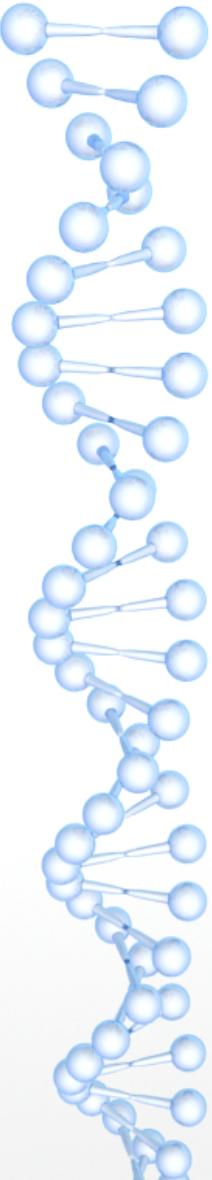
- Pošle naše commity do vzdialeného repozitára
- Ak nemáme nijaké nové dátá, sme o tom informovaní
- Môže skončiť neúspešne (povieme pri Merge-ovaní)



## 4.e) Fetch

*\$ git fetch*

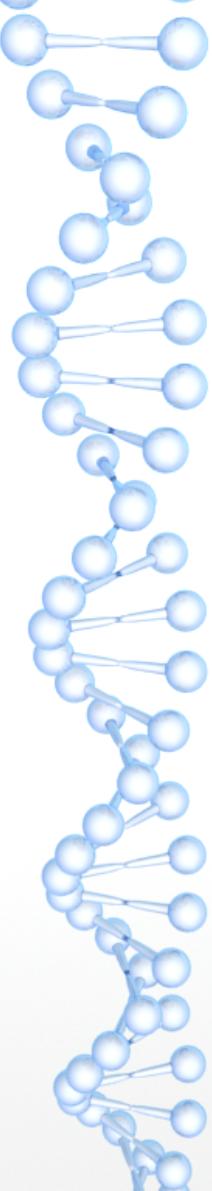
- Získá nové commity zo vzdialeného repozitára
- Nič nezmení u nás
- Nikdy nespôsobí „problémy“



## 4.f) Checkout

`$ git checkout {meno vety}`

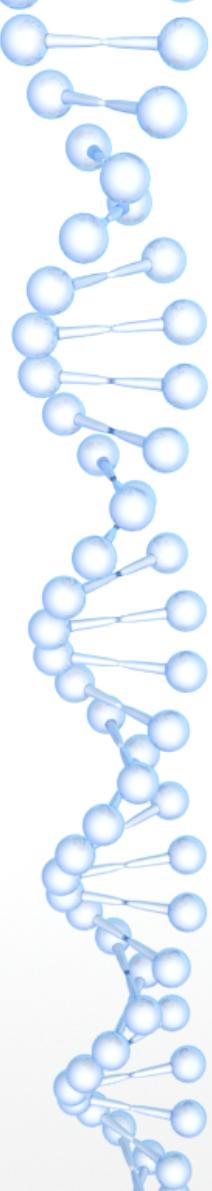
- Nikdy nespôsobuje komunikáciu so vzdialeným servrom
- Prepne nás na inú lokálne prístupnú vetu
- *Working dir* sa upraví podľa vety, na ktorú sa checkout-ujeme
- Môžeme prísť o dátu napr. vo *working dir* -zvykne sa (podľa prepínačov) opýtať, či to tak chceme



## 4.g) Pull

`$ git pull`

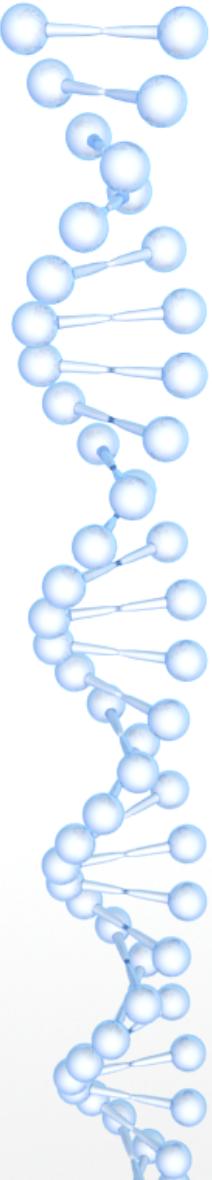
- Urobí niečo ako fetch (stiahnutie) a následný checkout na najnovší commit vo vzdialenom repozitári
- Tiež detekuje zmenu vo *working dir* -ak máme niečo necommitnuté, varuje nás
- **Jeho cieľom je čo naintelligentnejšie ZOSYNCHRONIZOVАŤ našu lokálnu a vzdialenú vetvu**
- Robí ešte skrytý Merge...



## 4.h) Merge-ovanie cez kdiff3

`$ git merge {iná vetva}`

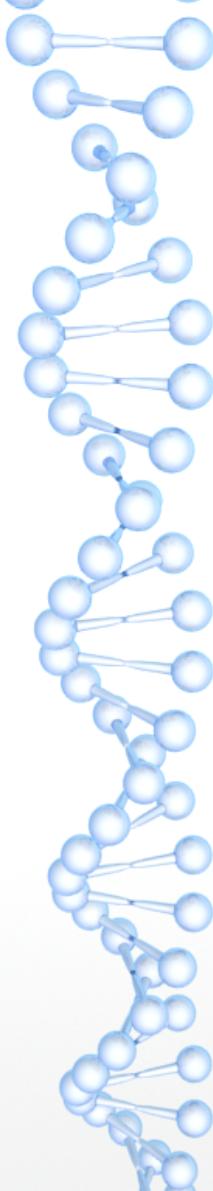
- Spojí aktuálnu a {inú vetvu}
- Musí riešiť (prípadné) **kolízie kódu** -triviálne prípady rieši sám; neautomatizovateľné necháva kóderovi =**konfliktný kód**
- Konfliktný kód prenecháva špecializovaným programom -napr. *kdiff3* (konfigurovatel'né)
- Je často najobávanejšou operáciou začiatočníkov



## 4.i) Stash

\$ *git stash*

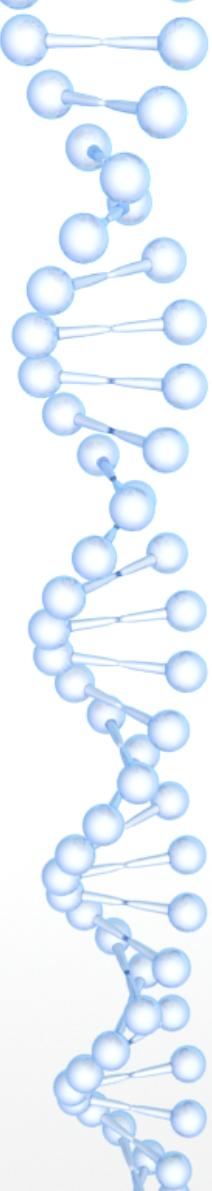
- Umožňuje narýchlo odsunúť aktuálne ešte necommittedé working dir zmeny
- Dá sa k nim hocikedy vrátiť cez \$ *git stash pop*
- Funguje na princípe zásobníka
- Vhodný pri operatívnych zásahoch  
(použiteľný na rýchle prepínanie sa medzi viac rozrobenými vecami)



4.j) Čo keď spravím chybu? (opravy bežných chýb)

$$\begin{aligned}
 c &= a + b + d \\
 c &\cdot (T \cdot S \cdot (\Omega \cdot 10^6) + 3\alpha + 2 \cdot 3 \ln(11))^2 \\
 c &\cdot (T \cdot S \cdot \log \frac{1}{S+P} + 3\alpha + 6 \ln(11))^2 \\
 c &\cdot \left[ \int_{x_1}^{x_2} \sum_{n=1}^{\infty} \alpha dx + \frac{3[(3+n)(1+6 \cdot 3^{\frac{n}{2}})]}{(5+y)(8+z)+1} + 6 \ln(11) \right]^2 \\
 c &\cdot \left[ \int_{x_1}^{x_2} \frac{(3+n)^2 \cdot (1+6 \cdot 3^{\frac{n}{2}})}{(5+y)(8+z)+1} dx + \frac{3[(3+n)^2 \cdot 6 \cdot 3^{\frac{n}{2}}]}{(5+y)(8+z)+1} + 6 \ln(11) \right]^2 \\
 c &\cdot \left[ \int_{x_1}^{x_2} \frac{(3+n)^2 \cdot ((\beta-180^\circ)+3^\circ)}{(5+y)(8+z)+1} dx + \frac{3[(3+n)^2 \cdot ((\beta-180^\circ)+3^\circ)]}{(5+y)(8+z)+1} + 6 \ln(11) \right]^2 \\
 c &\cdot \left[ \int_{x_1}^{x_2} \frac{\sqrt{3+n^2 + ((\beta-180^\circ)+3^\circ)^2}}{(45+y)(8+z)+1} dx + \frac{3\sqrt{3+n^2 + ((\beta-180^\circ)+3^\circ)^2}}{(45+y)(8+z)+1} + 6 \ln(11) \right]^2 \\
 c &= \sqrt{\left[ \int_{x_1}^{x_2} \sum_{n=1}^{\infty} \alpha dx + \frac{3[(3+n)(1+6 \cdot 3^{\frac{n}{2}}) + 6 \ln(11)]}{(5+y)(8+z)+1} + \log P \right]^2}
 \end{aligned}$$

$$\frac{\sqrt{2}}{2} = \sqrt{ }$$

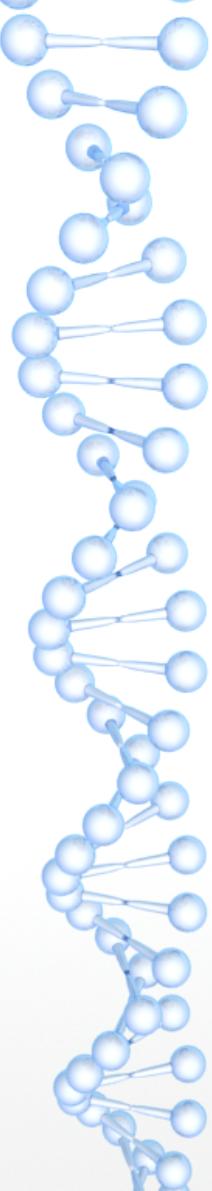


## 4.j) Čo ked' spravím chybu? (opravy bežných chýb)

- V commite som urobil pravopisnú chybu a zabudol súbor:

```
$ git add {zabudnutý súbor}
```

```
$ git commit –amend # pozmeni posledny commit
```

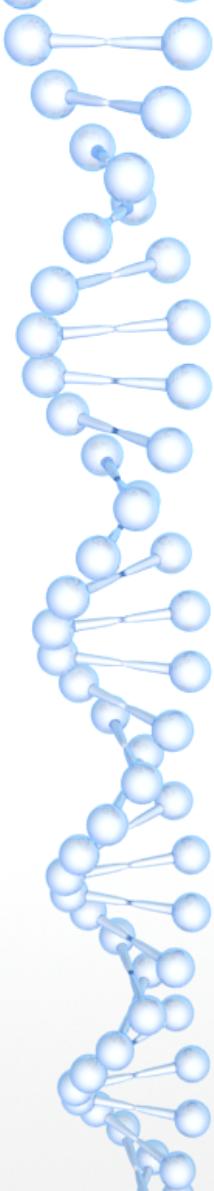


## 4.j) Čo ked' spravím chybu? (opravy bežných chýb)

- Posledný commit je CELÝ zle a chcem ho zrušiť (napr. aj pri zlom merge - chcem zrušiť merge)

```
$ git reset --hard HEAD^
```

```
$ git reset --soft HEAD^
```



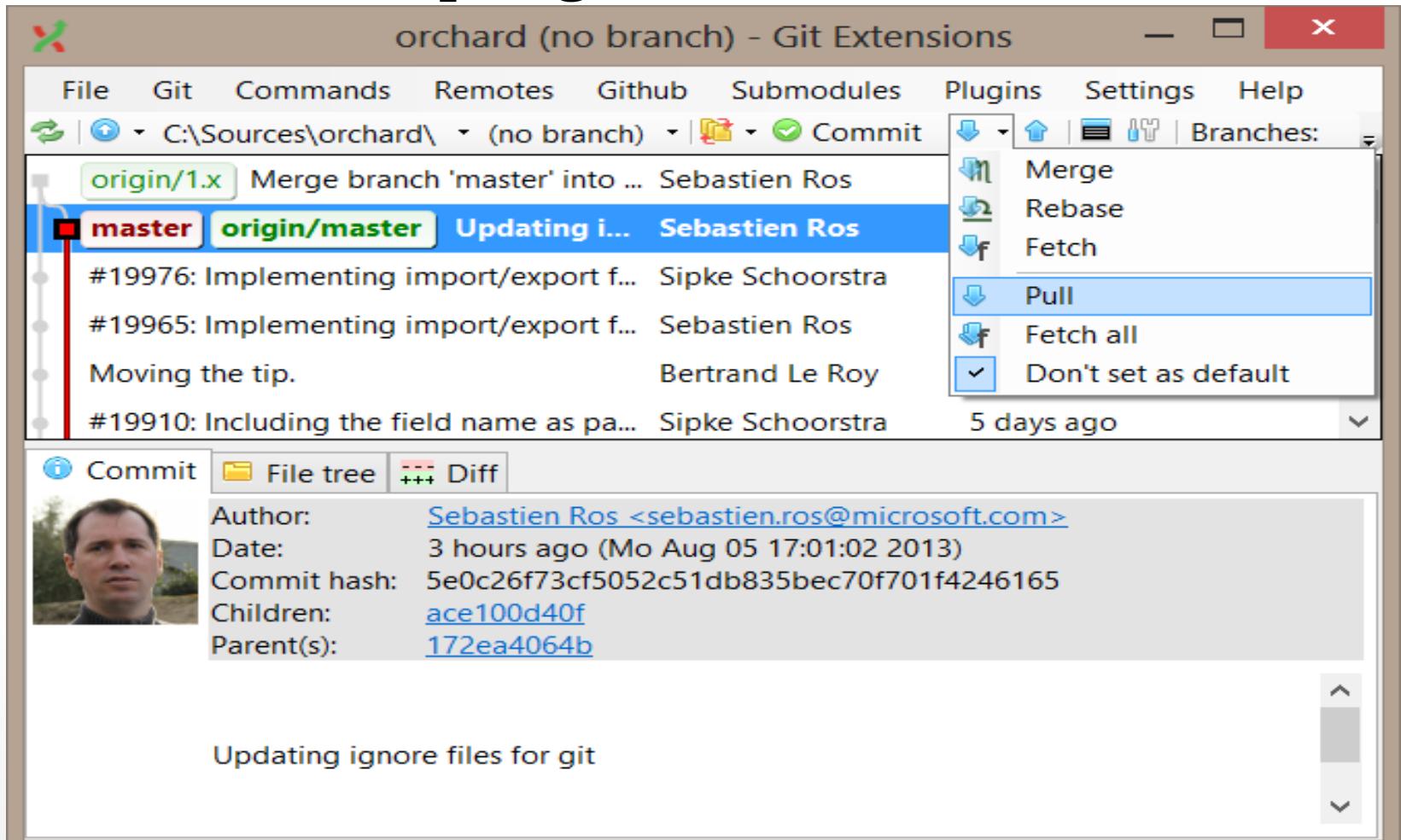
## 4.j) Čo ked' spravím chybu? (opravy bežných chýb)

- Ak už je push-nutý nejaký commit a chcem nad ním urobiť „undo“

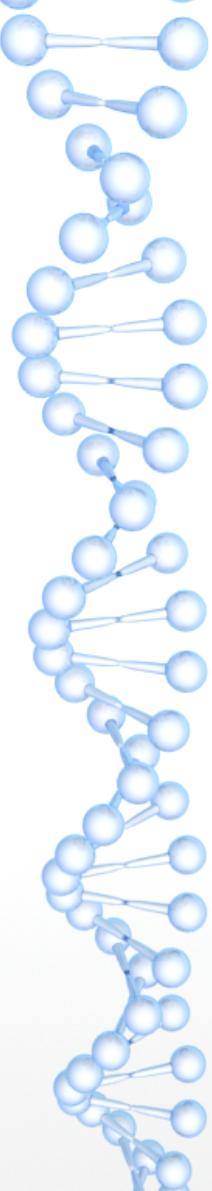
*\$ git revert {oznacenie zleho commitu - napr. Hash}*

- NIKDY nedeletujeme commity, ktoré sú už push-nuté! Niekoľko iných ich mohol pull-núť a zlý commit je v histórií jeho commitov. Vzdialený repozitár toto zmazanie preto zvyčajne nedovolí.

# 5. Rôzne GUI pre git -Windows -Git Extensions



# 5. Rôzne GUI pre git - MacOS - GitHub



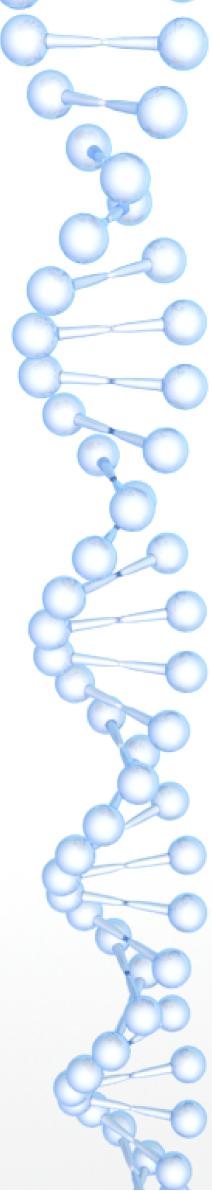
A screenshot of the GitHub desktop application interface on a Mac OS X system. The window title is "Repositories > ReactiveCocoa >". The sidebar on the left has icons for History, Changes, Branches, and Settings. The main area shows a list of commits in the "master" branch:

- 3515 commits in master (SHA: bba8fa0dee0e88ece51ed1cdc30b3dc5799af331)
- jshaber 5 hours ago: Merge pull request #864 from ReactiveCocoa/b35e733
- jspahrsummers 5 hours ago: Merge pull request #859 from ReactiveCocoa/1c39fc6
- jspahrsummers 6 hours ago: Test for +createSignal: disposing of inner subscriptions (bba8fa0) - This commit is selected.
- jspahrsummers 6 hours ago: Test that RACSerialDisposable.disposableWithBlock: returns nil if the inner signal is disposed (0549fcd)
- keithduncan 12 hours ago: Use `CFBridgingRetain` in place of a cast (c175133)
- jspahrsummers 2 days ago: Fix RACSerialDisposable leaking its set... (6de8f44)
- jspahrsummers 2 days ago: Dispose of RACSubscriber.disposable re... (d390f76)

The selected commit (bba8fa0) is expanded, showing the commit message "Test for +createSignal: disposing of inner subscriptions" and the author "Authored by jspahrsummers on Oct 14, 2013 at 10:18 AM. Verified to fail on master." Below the message, it says "Showing 1 changed file with 22 additions." and displays the diff for "ReactiveCocoaFramework/ReactiveCocoaTests/RACSignalSpec.m".

```
diff --git a/ReactiveCocoaFramework/ReactiveCocoaTests/RACSignalSpec.m b/ReactiveCocoaFramework/ReactiveCocoaTests/RACSignalSpec.m
@@ -223,6 +223,28 @@ describe:@"subscribing", ^{
224     expect(currentScheduler).willNot.beNil();
225 };
226 +
227 + it(@"should automatically dispose of other subscriptions from
228 + +createSignal:", ^{
229 +     __block BOOL innerDisposed = NO;
230 +
231 +     RACSignal *innerSignal = [RACSignal
232 + createSignal:^(id<RACSubscriber> subscriber) {
233 +         return [RACDisposable disposableWithBlock:^{
234 +             innerDisposed = YES;
235 +         }];
236 +
237 +         RACSignal *outerSignal = [RACSignal createSignal:^{
238 +             RACDisposable * (id<RACSubscriber> subscriber) {
239 +                 [innerSignal subscribe:subscriber];
240 +                 return nil;
241 +             ];
242 +         }];
243 +     }];
244 + }
```

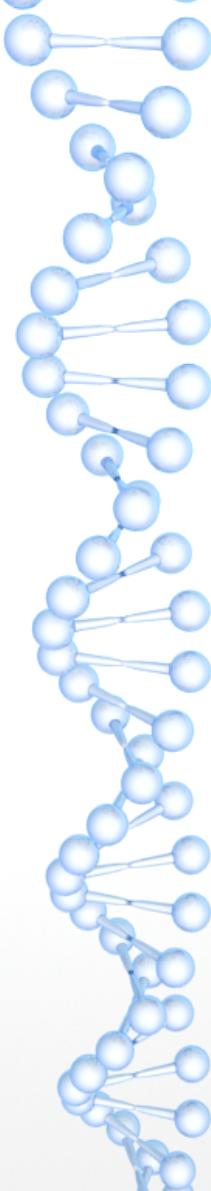
# 5. Rôzne GUI pre git -Linux -qgit



The image shows a screenshot of the qgit graphical user interface. The interface has a menu bar with File, Edit, View, Actions, and Help. A toolbar above the main window includes icons for file operations like Open, Save, and Commit, along with a search bar labeled "Short log". The main window is divided into several panes:

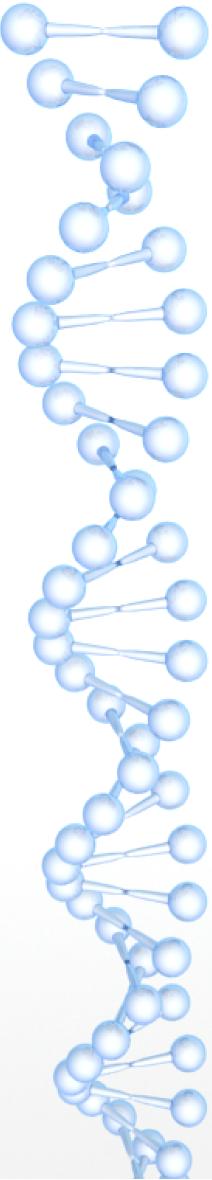
- Git tree:** On the left, it shows the project's directory structure with files like deaf, core, dbDumps, logs, static, .gitignore, countries.json, data\_class\_m..., howToStart.txt, langs.json, lessons-data..., LICENSE, manage.py, nepocujuci\_ci..., nepocujuci\_ci..., nepocujuci.xmi, README.md, and requirements.txt.
- Rev list:** This pane contains a list of commits. The first commit is highlighted in orange and labeled "Working directory changes". Subsequent commits are listed with their hash codes (e.g., v0.123, v0.122, v0.121, v0.120, v0.119) and descriptions:
  - v0.123 fix Feature #2209
  - v0.122 Feature #2209
  - v0.121 removed required params for UserSet method
  - v0.120 fixed security -using requests  
changed getting videoSize through requests module  
updated token and validation way  
added requests as dependency
  - v0.119 less verbose methods for data classes  
SECURING the external https call (FB API, iTunes); fixed DictionarySe...
- Short Log:** Below the rev list, there is a detailed log of the selected commit (v0.122). It shows the commit message, the author (Babbbrak Poradny), and the date (27.1.2014 17:38). The log also includes the diff between the previous and current state of the file.
- Diff:** On the right, there is a side-by-side diff viewer showing the changes made in the file.

At the bottom of the interface, a status bar displays the tag "Tag: v0.122 [Feature #2209]".

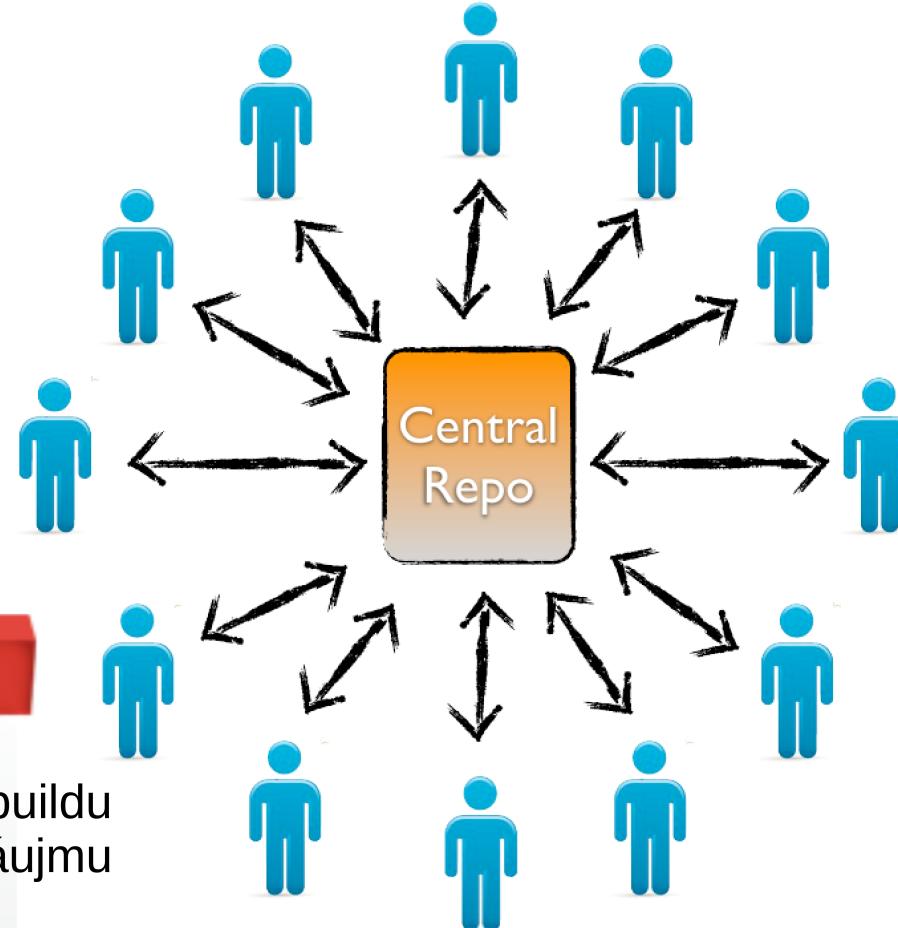


## 6. „Tvar“ developerskej komunity

- **Centrálny repozitár** (rovný s rovným) -zvykne byť pri closed source repozitároch
- **Diktátor** (niekto zaraďuje kód do hlavného stromu) -bežné pri OpenSource projektoch; ak sa ukáže diktátor zlý, urobí sa fork =iný „diktátor“ si založí vlastnú vetvu
- Vždy môže byť s **Mirroringom** (server má svoj mirror - podobné CDN / cloutu) -ponúka napr. SourceForge; lepšia geografická distribúcia
- Existuje aj možnosť miešaného closed +open vývoja

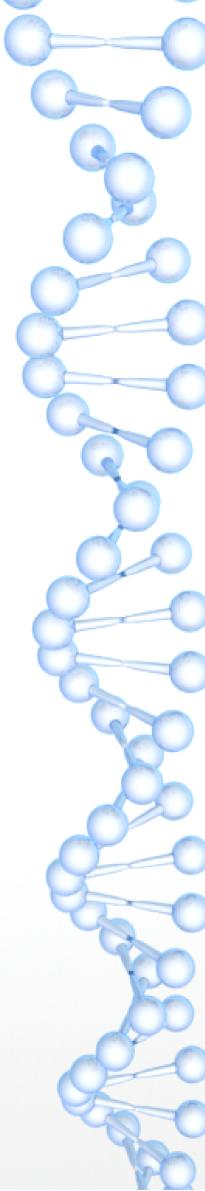


## 6. „Tvar“ dev komunity -centrálny repozitár

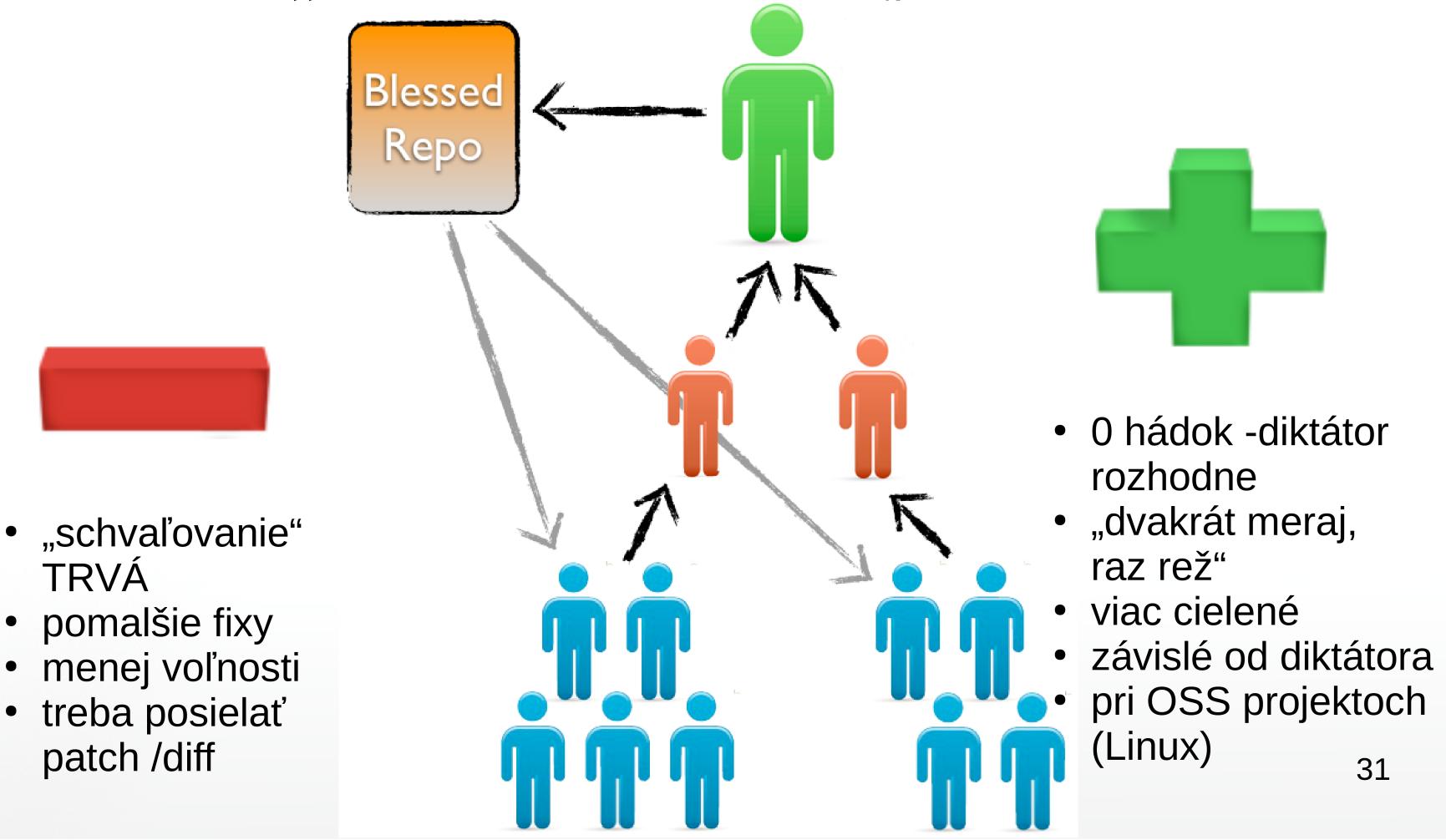


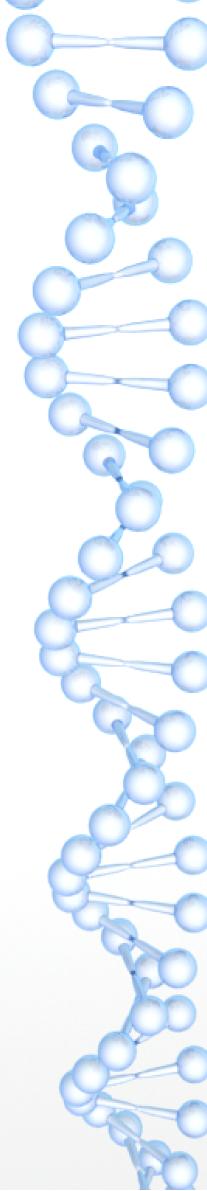
- stabilita buildu
- kolízie záujmu

- rýchlosť
- jednoduchosť
- v TB internet banking

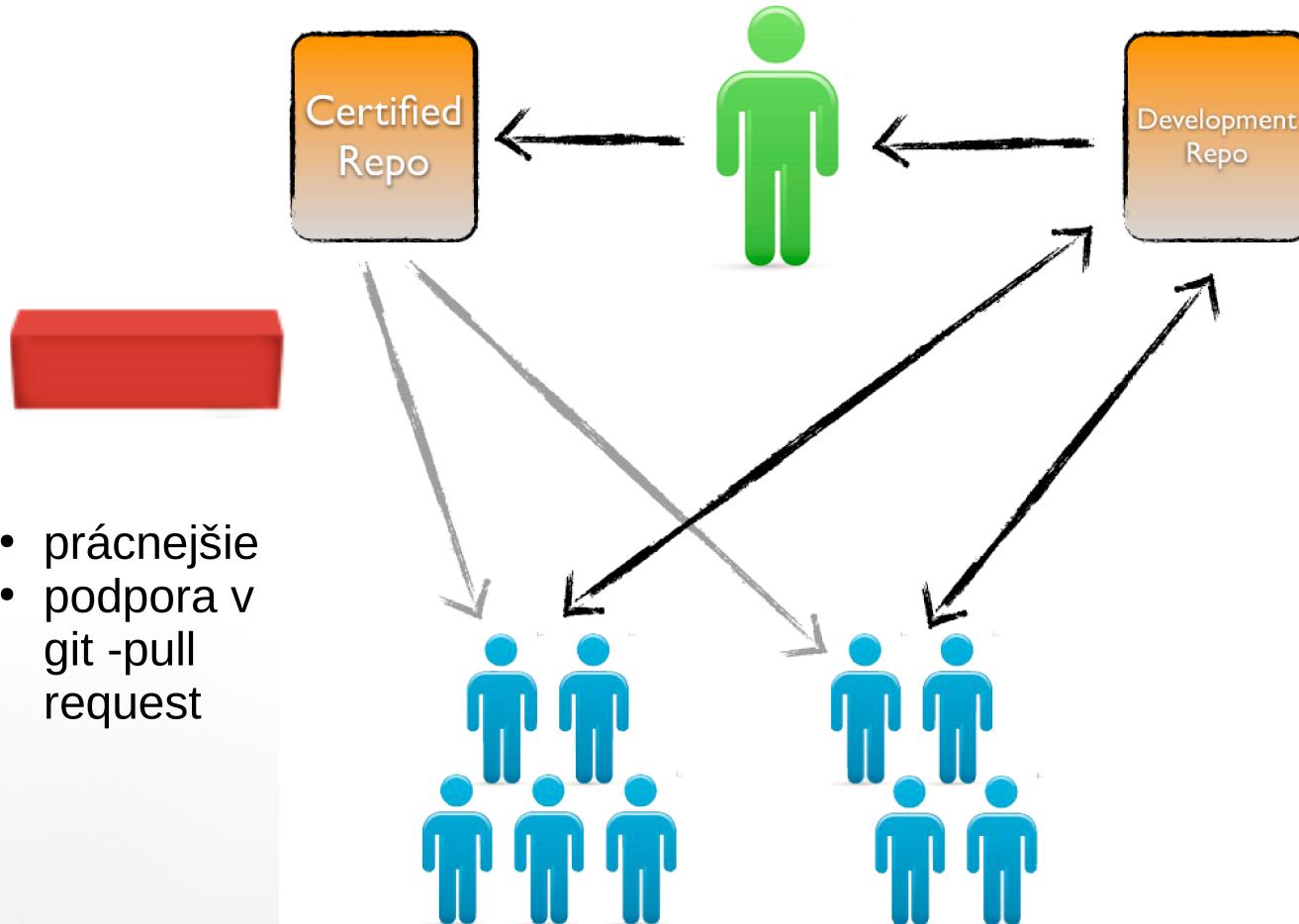


## 6. „Tvar“ dev komunity -diktátor



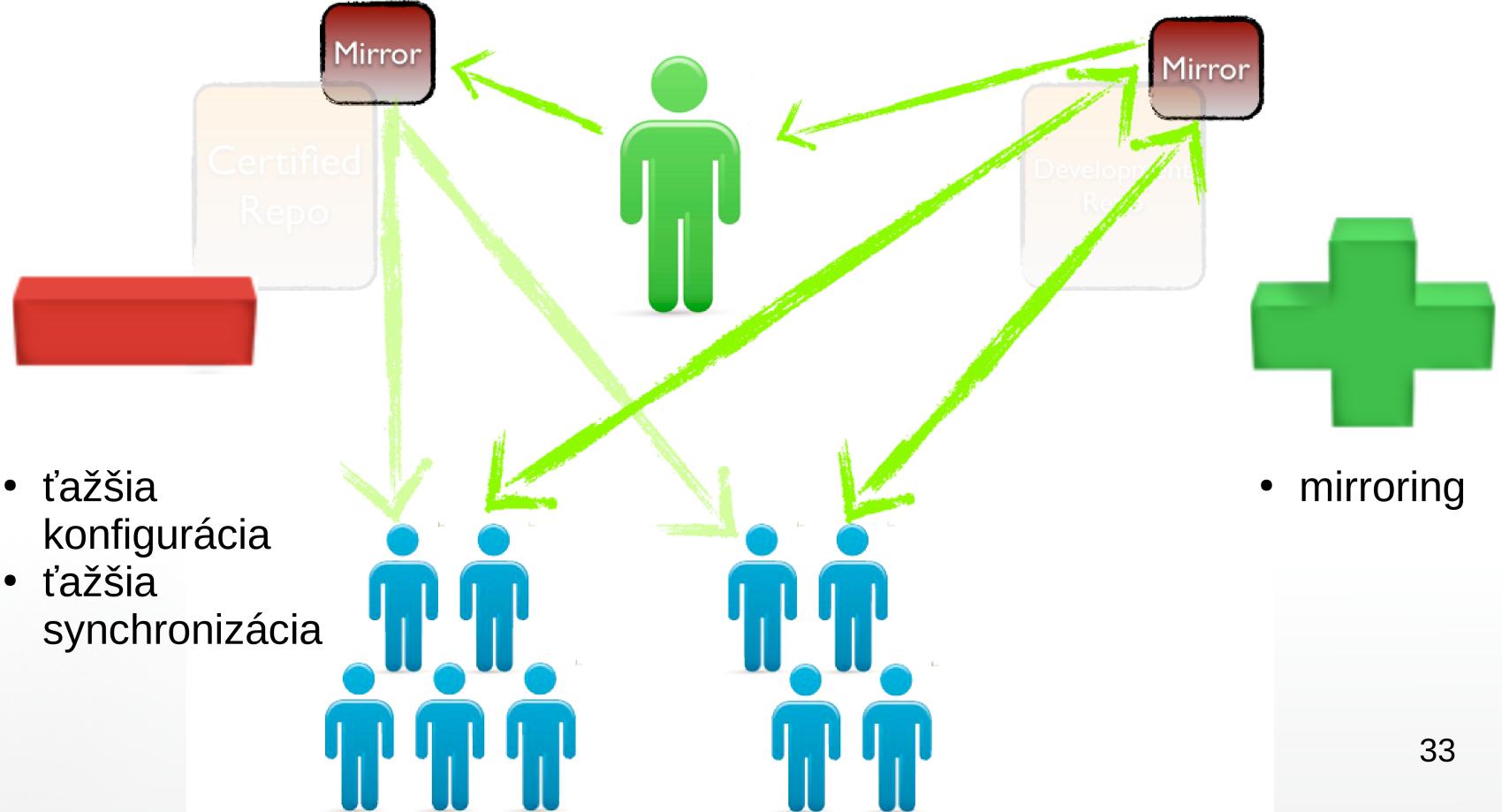


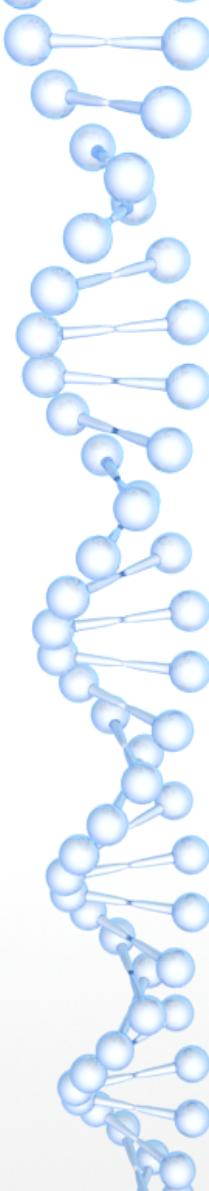
## 6. „Tvar“ dev komunity -diktátor v2



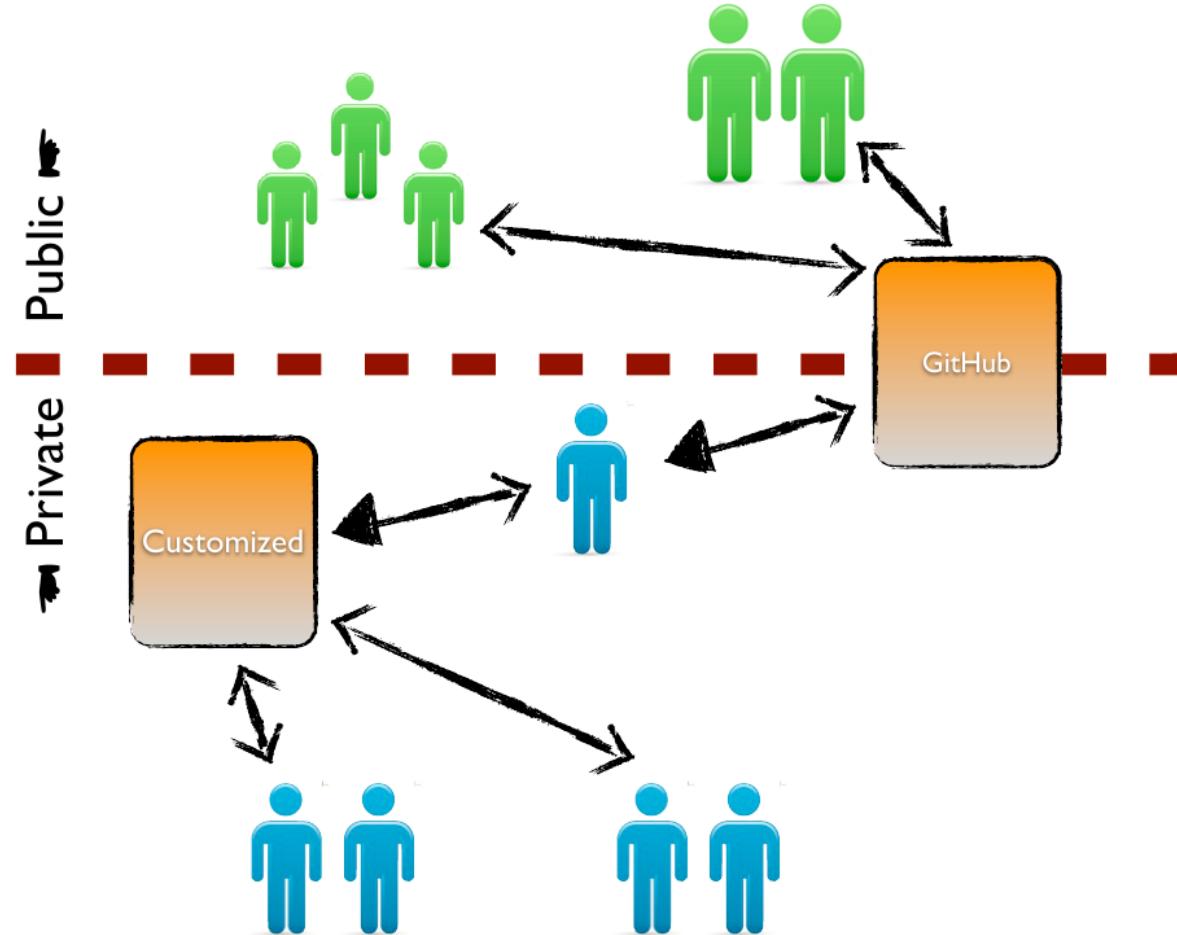
- certifikovaný repo je „čistý“

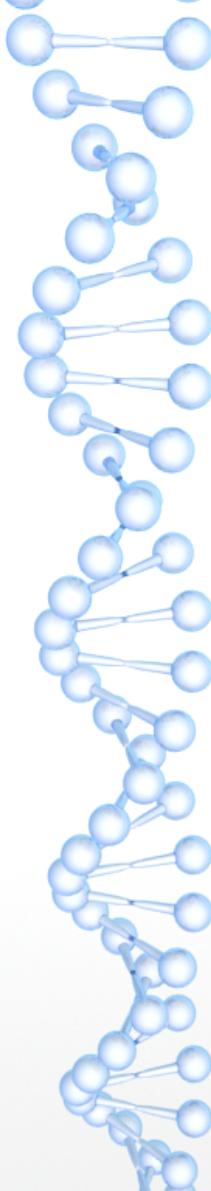
# 6. „Tvar“ dev komunity -diktátor v2 +mirror





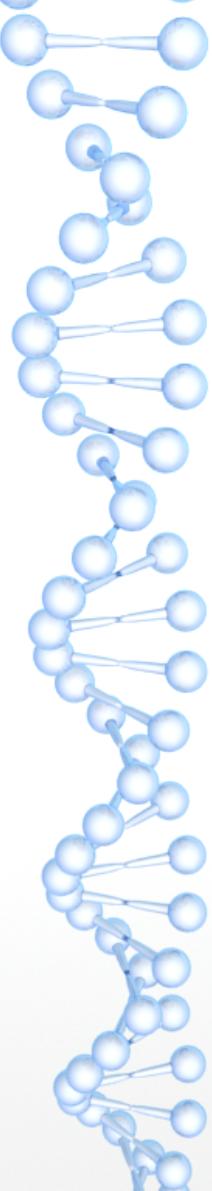
## 6. „Tvar“ dev komunity -open +closed





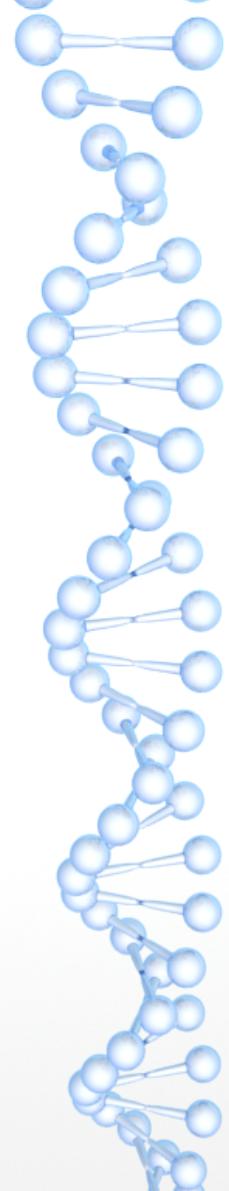
## 6. Sumár (alebo „Čo už vieme“)

1. V čom je git dobrý
2. Ako vnútorne funguje GIT
3. Ako vytvoriť a naklonovať GIT repozitár
4. Ako komunikovať so vzdialeným repozitárom
5. Ako commitnúť zmeny, pridať súbor, merge-ovať vetvy
6. Ako napraviť najzvyčajnejšie chyby
7. Ktorý GUI nástroj je fajn

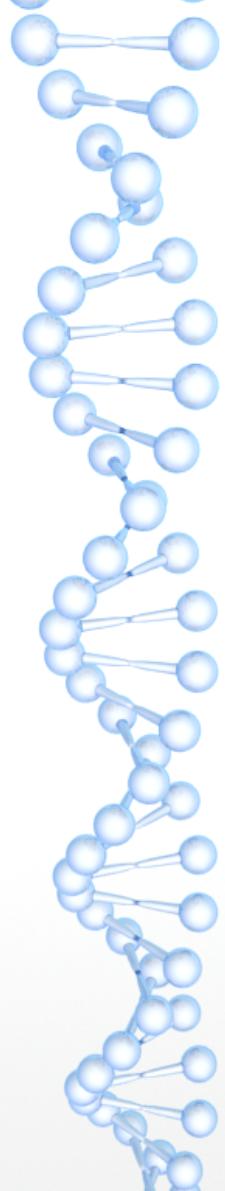


## 7. Záverom

- Najvhodnejšia príručka ku GIT je *Pro GIT* od Scotta Chacona: je v češtine, používa veľa obrázkových príkladov, ľahko čitateľná, obsahuje takmer celú user dokumentáciu GIT-u
- Git dokáže robiť len *automatizovateľnú* prácu - nemožno od neho čakať „telepatiu“ (vid' Merge-ovanie)
- GUI nástroje sú fajn, ale vždy je lepšie vidieť im na prsty



Otázky?



Ďakujem za pozornosť!