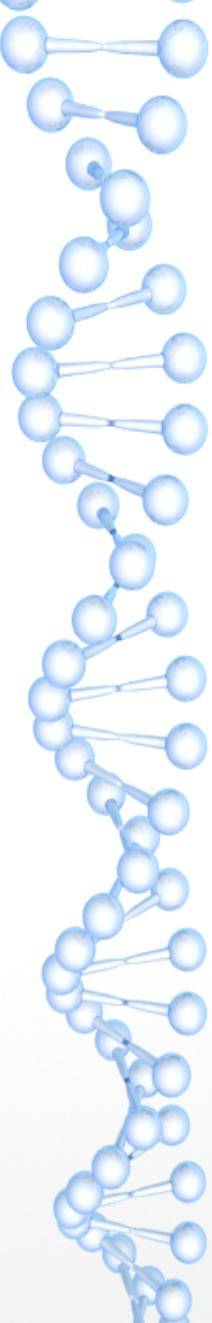


# GIT

## GIT primer

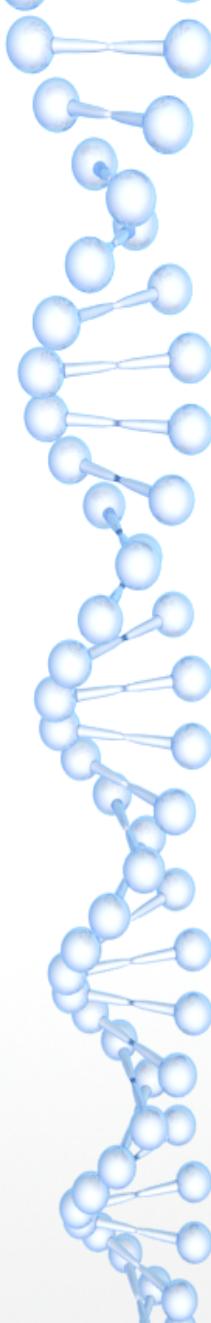
*Ako na GIT aj pre  
úplných začiatočníkov*

Miloš Korenčiak (MojaKomunita)



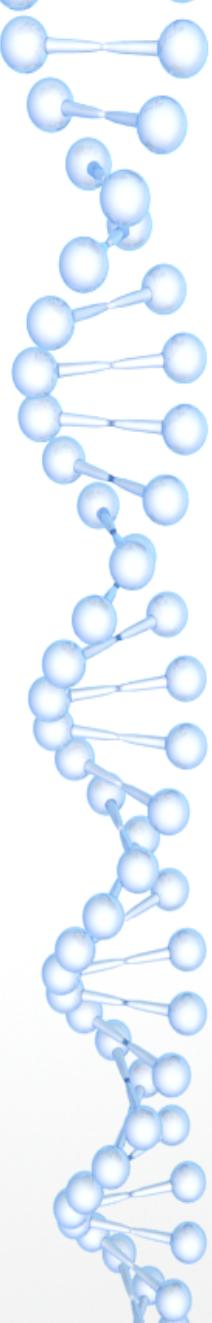
# Obsah

- 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 zistit', či sa projekt stíha

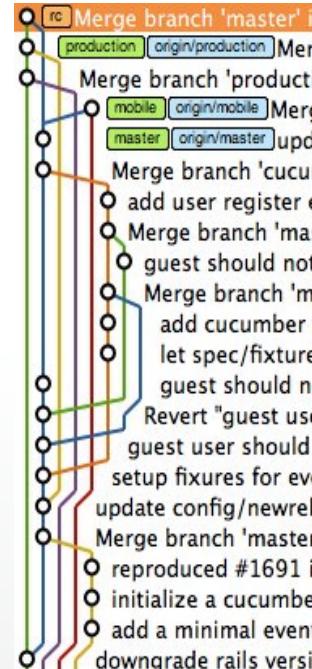
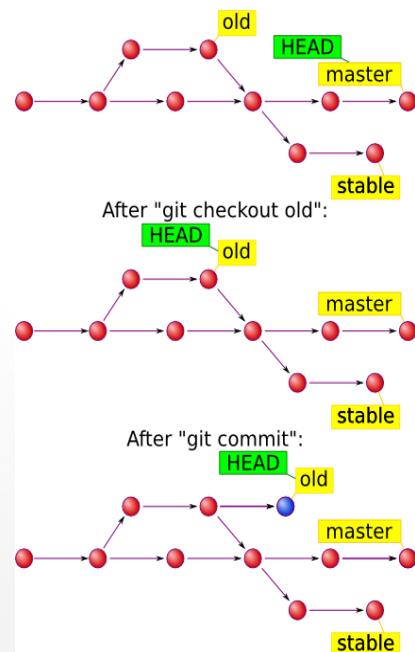


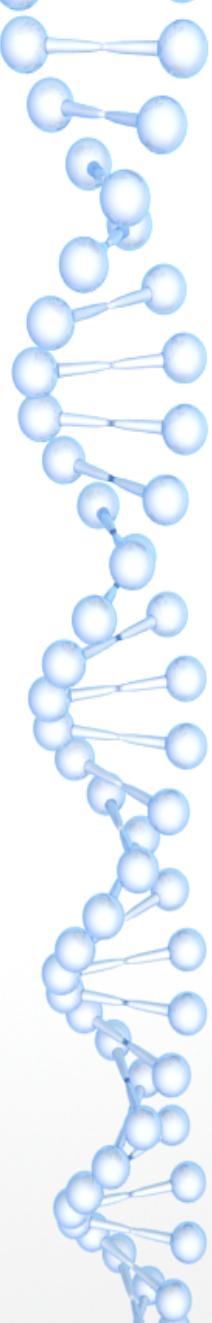
## 2. Čo je vlastne GIT zač?

- *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

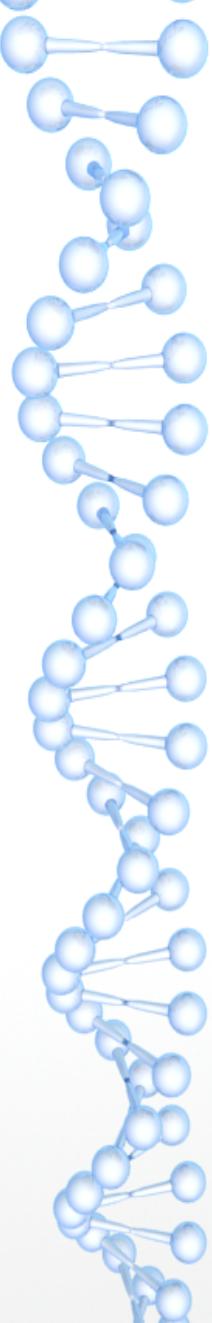




## 2. Čo je vlastne GIT zač?

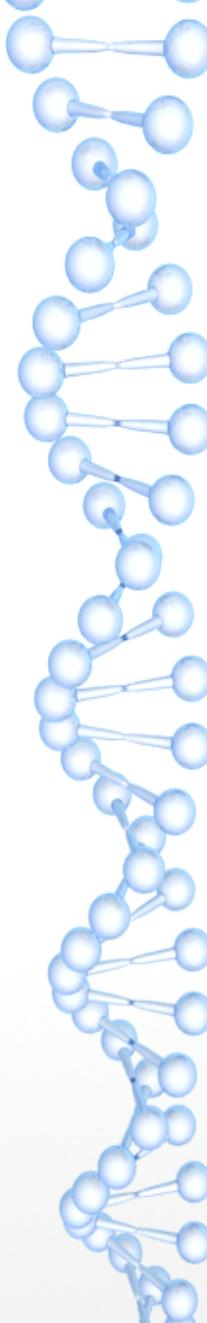
**Software tools for revision control are essential for the organization of multi-developer projects.**

([Wikipedia](#))



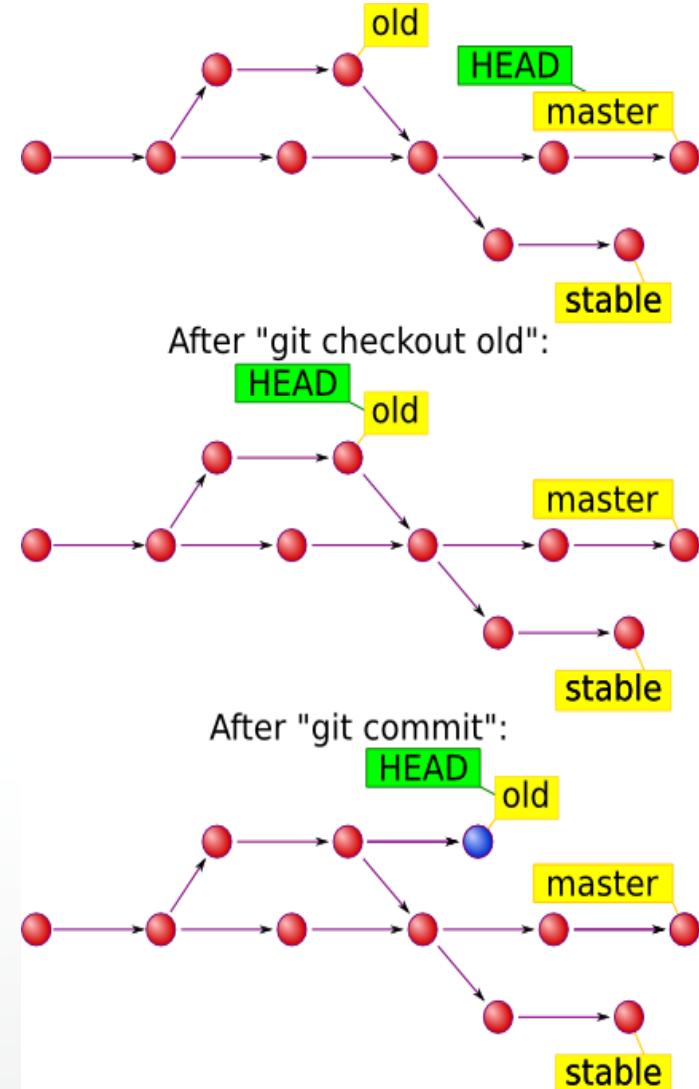
### 3. Tri stromy/databázy v GIT -working dir

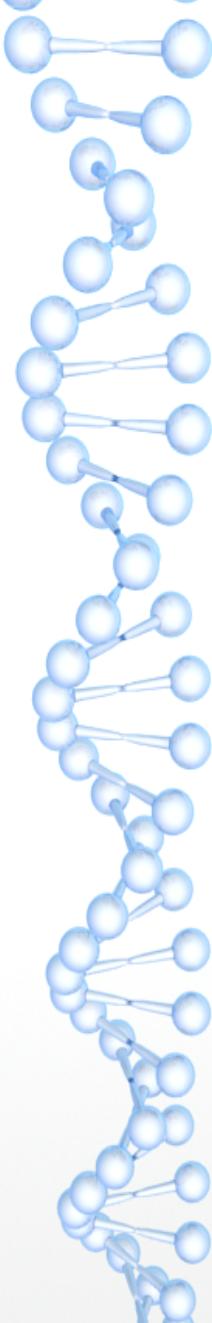
- Ten adresár v ktorom priamo programujeme
- Je v ňom teda fyzicky každý súbor projektu
- GIT prehľadáva aktuálny adresár a rekurzívne aj všetky podadresáre
- Zistuje, ktoré súbory oproti poslednému commitu pribudli (a potenciálne by sa mali pridať do repozitára), a ktoré súbory v repozitári (tzv. verzované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



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

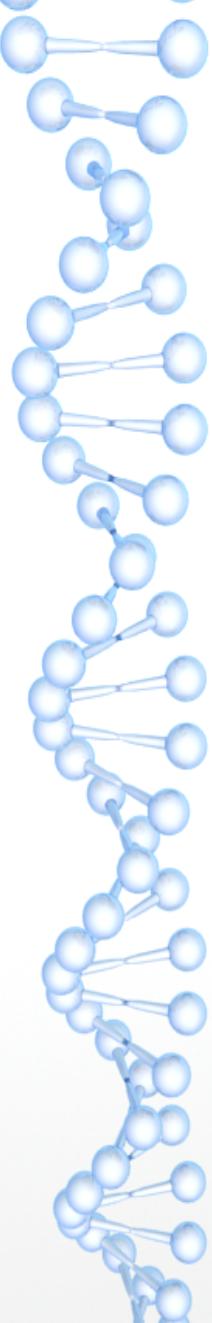
- Je to posledný commit v aktuálnej vetve „stromu“ (strom je celý GIT repozitár)
- Ak urobíme commit, stane sa synom aktuálneho HEAD commitu; HEAD ukazovateľ sa potom posunie na náš najnovší commit
- Ukazuje kde pracujeme





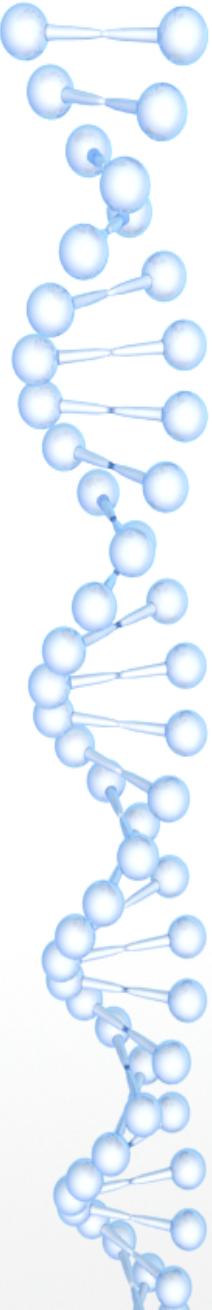
### 3. Tri stromy/databázy v GIT -index

- 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

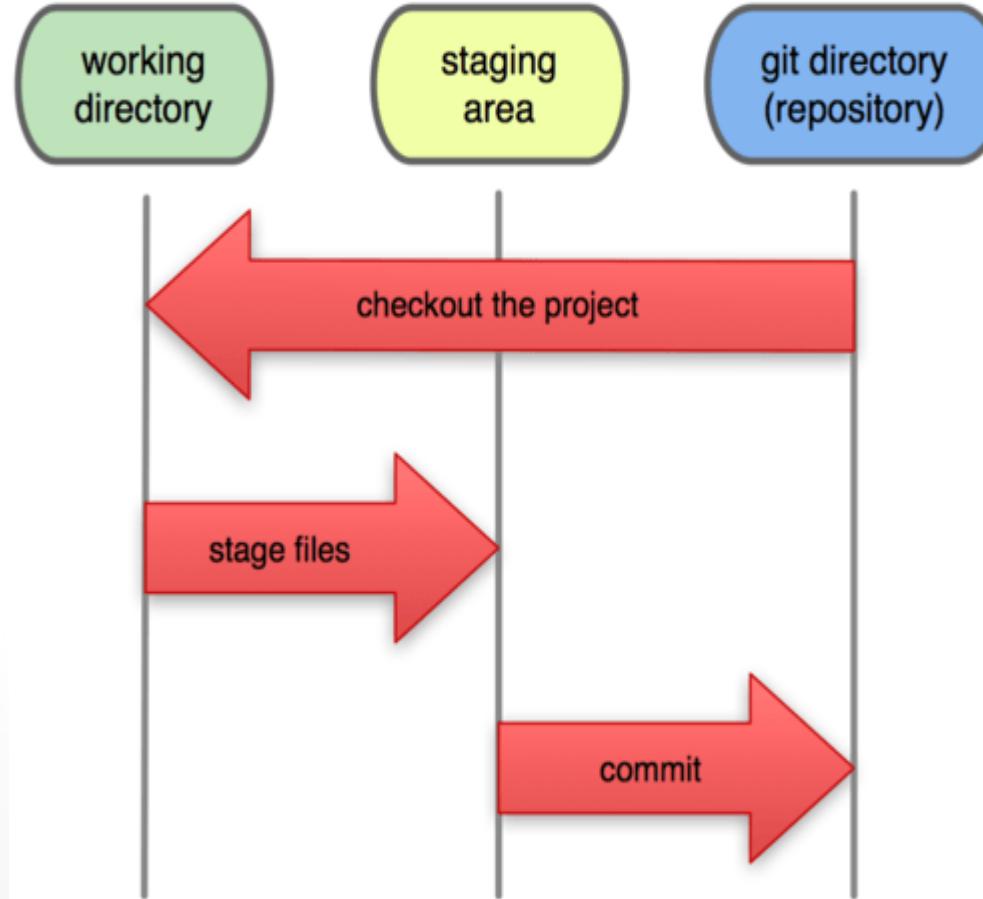
- Programovaním meníme working dir
- Príkazom git add (bude ukázaný neskôr) prenášame working dir zmeny do indexu
- Príkazom git commit (neskôr) prenášame index alebo aj working dir do stromu repozitára (až ten je verzovaný)
- IBA strom GIT repozitára je schopný priamo komunikovať so vzdialenosťmi GIT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

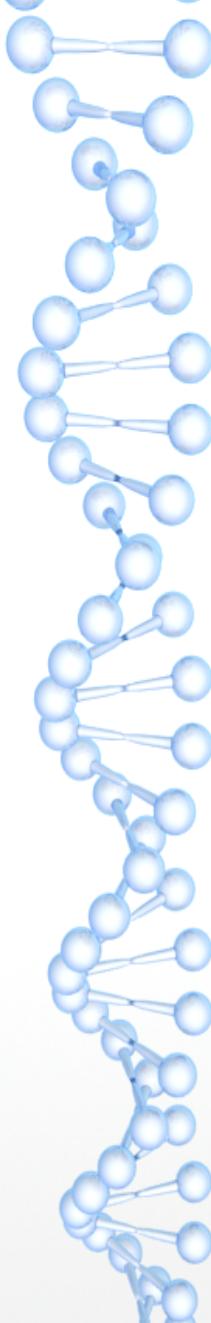


# 3. Tri stromy/databázy v Git -ich

## spoluhraná

### Local Operations

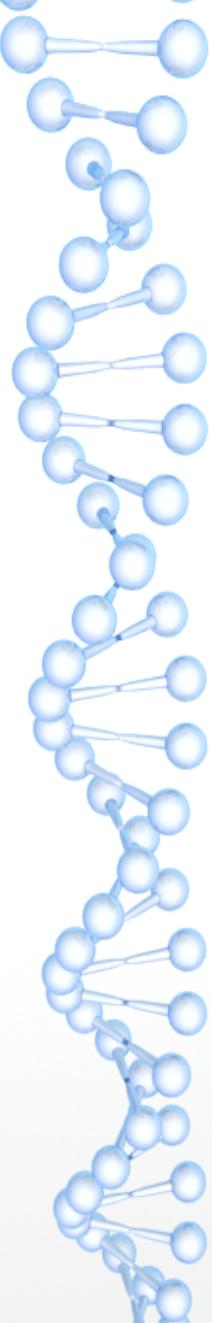




## 4. Hands on lab

(praktická ukážka v *git bash* konzole)

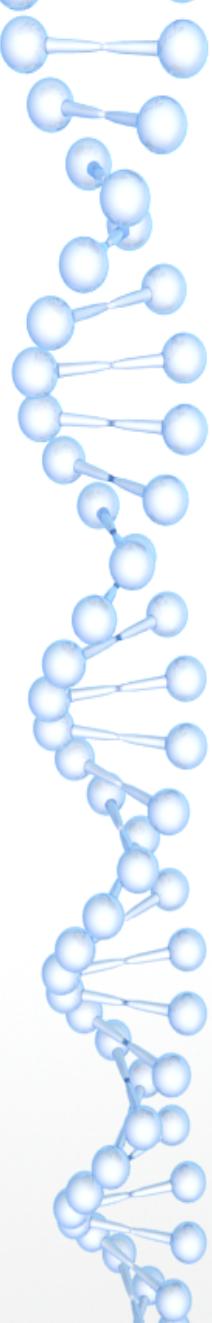
Predved'me si to na jednoduchom  
projekte ...



## 4.a) Vytvorenie repozitára

*\$ git init*

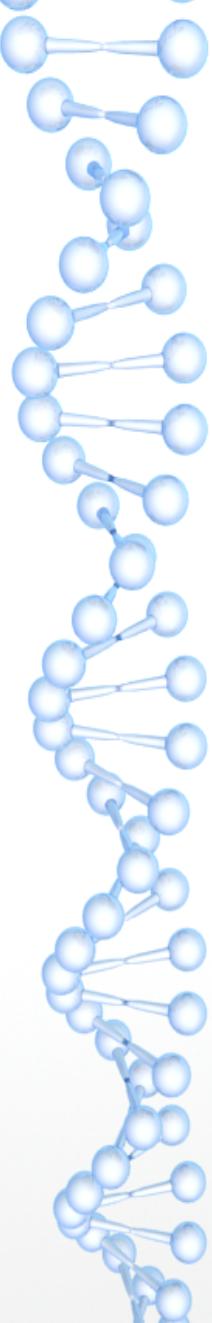
- Vytvorí v aktuálnom adresári lokálny git repozitár
- Jeho všetky dáta sú v jedinom *.git* skrytom podadresári



## 4.b) Stiahnutie repozitára

```
$ git clone https://github.com/vishnevskiy/battlenet.git  
$ git clone /home/m/kodenie/effective-django-tutorial
```

- 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*

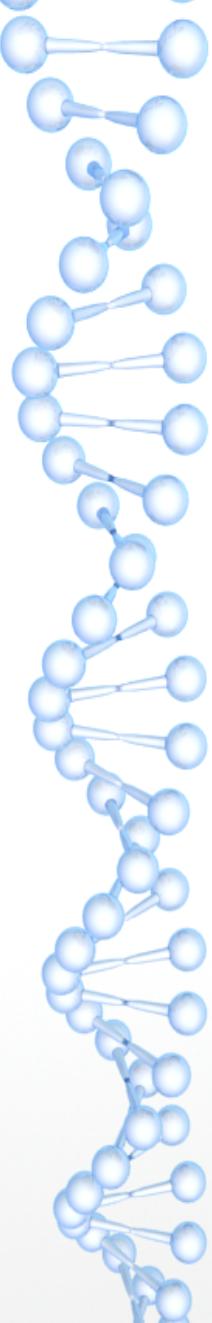
*\$ git add haha.txt*

*\$ git commit*

*\$ git commit -a*

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

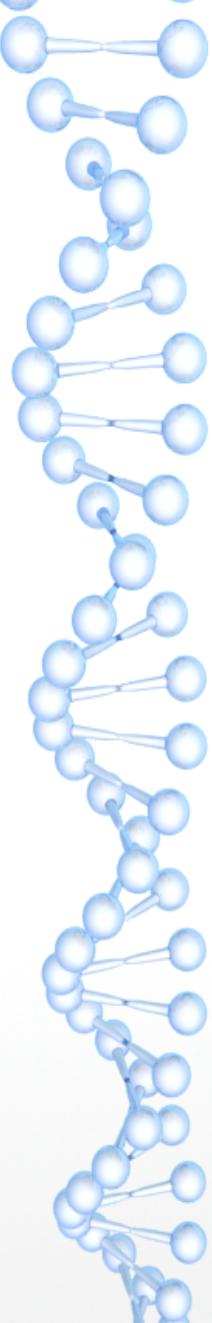
- Preskúmame stav repozitára
- Pridáme súbor, aby sa oň git staral
- Commit-neme jeho zmeny do git-u („Tri stromy git-u“)



## 4.d) Push

*\$ git push*

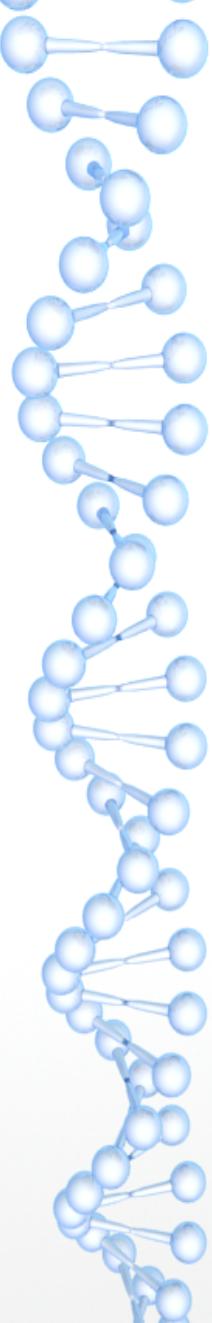
- Pošle naše commity do vzdialeného repozitára
- Ak nemáme nijaké nové dátá, sme o tom informovaní
- Môže skončiť neúspešne (povieme pri Merge-ovani)



## 4.e) Fetch

*\$ git fetch*

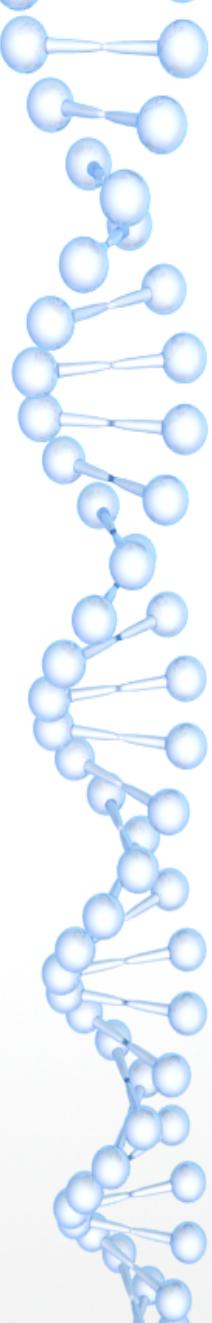
- Získa nové commity zo vzdialeného repozitára
- Nič nezmení u nás
- Nikdy nespôsobí „problémy“



## 4.f) Checkout

`$ git checkout {meno vety}`

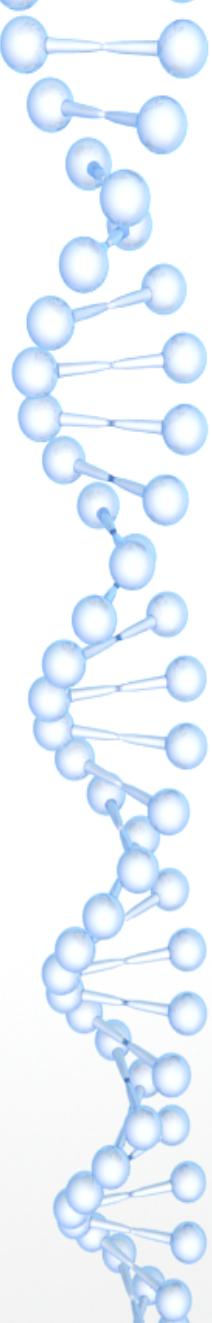
- Nikdy nespôsobuje komunikáciu so vzdialeným servrom
- Prepne nás na inú lokálne prístupnú vetvu
- *Working dir* sa upraví podľa vety, na ktorú sa checkout-ujeme
- Môžeme prísť o dátu napr. vo *working dir* -zvykne sa (podľa prepínačov) opýtať, či to tak chceme



## 4.g) Pull

\$ *git pull*

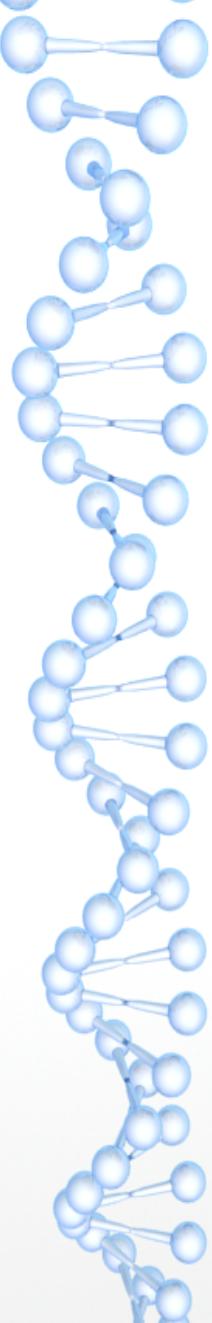
- Urobí niečo ako fetch (stiahnutie) a následný checkout na najnovší commit vo vzdialenom repozitári
- Tiež detektuje zmenu vo *working dir*-ak máme niečo necommitnuté, varuje nás
- **Jeho cieľom je čo nainteligentnejšie ZOSYNCHRONIZOVАŤ našu lokálnu a vzdialenú vetvu**
- Robí ešte skrytý Merge...



## 4.h) Merge-ovanie cez kdiff3

*\$ git merge {iná vetva}*

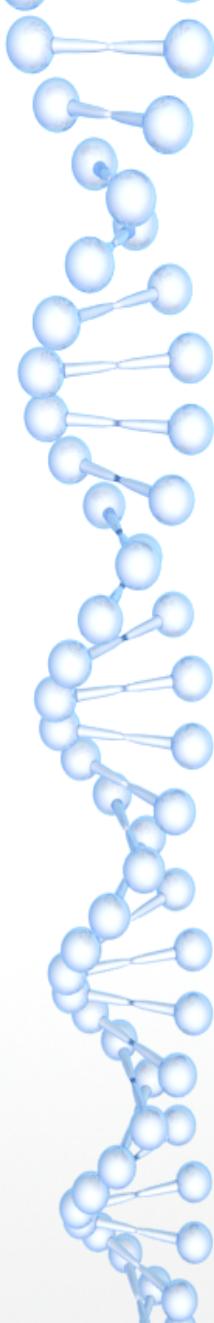
- Spojí aktuálnu a {inú vetvu}
- Musí riešiť (prípadné) **kolízie kódu** -triviálne prípady rieši sám; neautomatizovateľné necháva kóderovi =**konfliktný kód**
- Konfliktný kód prenecháva špecializovaným programom -napr. *kdiff3* (konfigurovateľné)
- Je často najobávanejšou operáciou začiatočníkov



## 4.i) Stash

*\$ git stash*

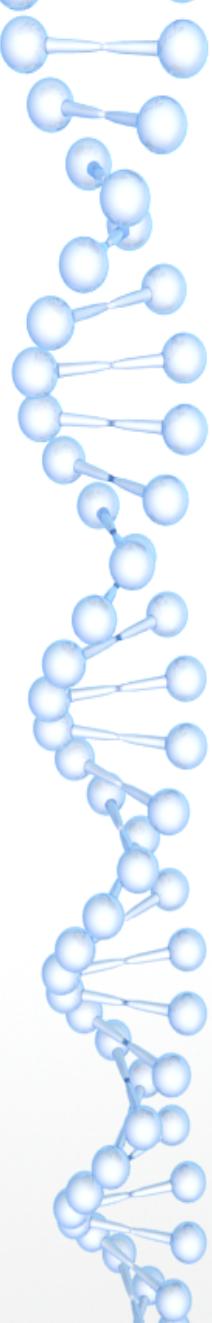
- Umožňuje narýchlo odsunúť aktuálne ešte necommit-nuté working dir zmeny
- Dá sa k nim hocikedy vrátiť cez *\$ git stash pop*
- Funguje na princípe zásobníka
- Vhodný pri operatívnych zásahoch  
(použiteľný na rýchle prepínanie sa medzi viac rozrobenými vecami)



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

$$\begin{aligned}
 C &= \alpha + \beta + \gamma \\
 C &\cdot (T \cdot S \cdot (\Omega - 10^\circ) + 3\alpha + 2 \cdot 3 \ln(11)) \\
 C &\cdot (T \cdot S \cdot \log \frac{\beta}{\alpha + \beta} + 3\alpha + 6 \ln(11))^2 \\
 C &\cdot \left[ \int_{x_1}^{x_2} \alpha dx + \frac{3[(3+7)x] + [6+3\pi]}{(5+\gamma)(8+2)+1} + 6 \ln(11) \right]^2 \\
 C &\cdot \left[ \int_{x_1}^{x_2} \frac{(3+7)x + [6+3\pi]}{(5+\gamma)(8+2)+1} dx + \frac{3[(3+7)x] + [6+3\pi]}{(5+\gamma)(8+2)+1} + 6 \ln(11) \right]^2 \\
 C &\cdot \left[ \int_{x_1}^{x_2} \frac{(3+7x)^2 + (\beta - 180^\circ) + 3\pi}{(5+\gamma)(8+2)} dx + \frac{3[(3+7)x] + [(\beta - 180^\circ) + 3\pi]}{(5+\gamma)(8+2)+1} + 6 \ln(11) \right]^2 \\
 C &\cdot \left[ \sum_{x_1}^{x_2} \frac{\sqrt{3+7x} + (\beta - 180^\circ) + 3\pi}{10\Omega + 6\pi - 1} dx + \frac{3\sqrt{3+7x} + (\beta - 180^\circ) + 3\pi}{10\Omega + 6\pi - 1} + \log \beta \right]^2 \\
 C &\cdot \boxed{\left[ \sum_{x_1}^{x_2} \frac{[(3\theta - 7x + (\beta - 180^\circ) + 3\pi)]^2}{(5\gamma + 1)(8 + 2)} \right] \frac{1}{10\Omega + 6\pi - 1} + \log \beta} \\
 C &= \boxed{\sqrt{\left[ \sum_{x_1}^{x_2} \frac{[(3\theta - 7x + (\beta - 180^\circ) + 3\pi)]^2}{(5\gamma + 1)(8 + 2)} \right] \frac{1}{10\Omega + 6\pi - 1} + \log \beta}}
 \end{aligned}$$

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

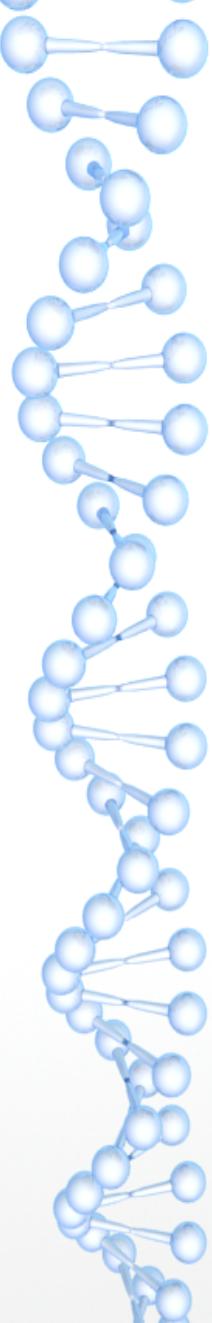


## 4.j) Čo ked' spravím chybu? (opravy bežných chýb)

- V commitе som urobil pravopisnú chybu a zabudol súbor:

```
$ git add {zabudnutý súbor}
```

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

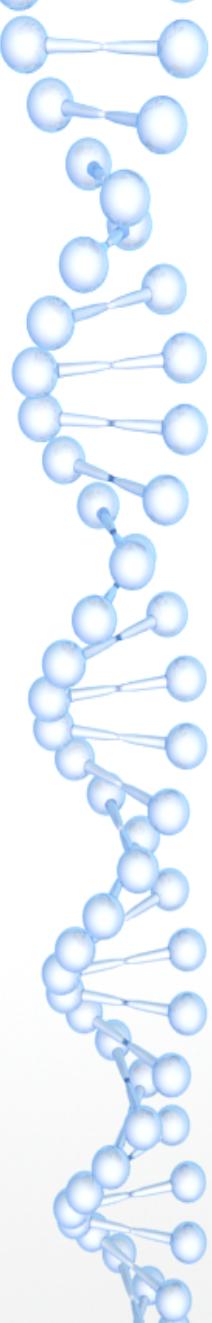


## 4.j) Čo ked' spravím chybu? (opravy bežných chýb)

- 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 urobit „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

orchard (no branch) - Git Extensions

File Git Commands Remotes Github Submodules Plugins Settings Help

C:\Sources\orchard\ (no branch) Commit

origin/1.x Merge branch 'master' into ... Sebastien Ros

**master origin/master Updating i... Sebastien Ros**

#19976: Implementing import/export f... Sipke Schoorstra

#19965: Implementing import/export f... Sebastien Ros

Moving the tip. Bertrand Le Roy

#19910: Including the field name as pa... Sipke Schoorstra 5 days ago

Commit File tree Diff

Author: [Sebastien Ros <sebastien.ros@microsoft.com>](mailto:Sebastien.Ros@microsoft.com)  
Date: 3 hours ago (Mo Aug 05 17:01:02 2013)  
Commit hash: 5e0c26f73cf5052c51db835bec70f701f4246165  
Children: [ace100d40f](#)  
Parent(s): [172ea4064b](#)

Updating ignore files for git

Merge  
Rebase  
Fetch  
Pull  
Fetch all  
Don't set as default

26

# 5. rôzne GUI pre git -Windows -GitHub

local



github



niik



reactiveui

The screenshot shows the GitHub desktop application interface. At the top, there are buttons for 'create', 'refresh', and 'tools'. Below that is a search bar labeled 'Filter Repositories'. The main area displays a list of repositories under the 'reactiveui' organization. The first repository, 'reactiveui/ReactiveUI', is highlighted with a blue background. Other listed repositories include 'reactiveui/ReactiveUI.Samples', 'reactiveui/RxUIProjectTemplate', and 'reactiveui/RxUI\_QCon'. Each repository entry has a small icon, a repository name, and a 'View' button.

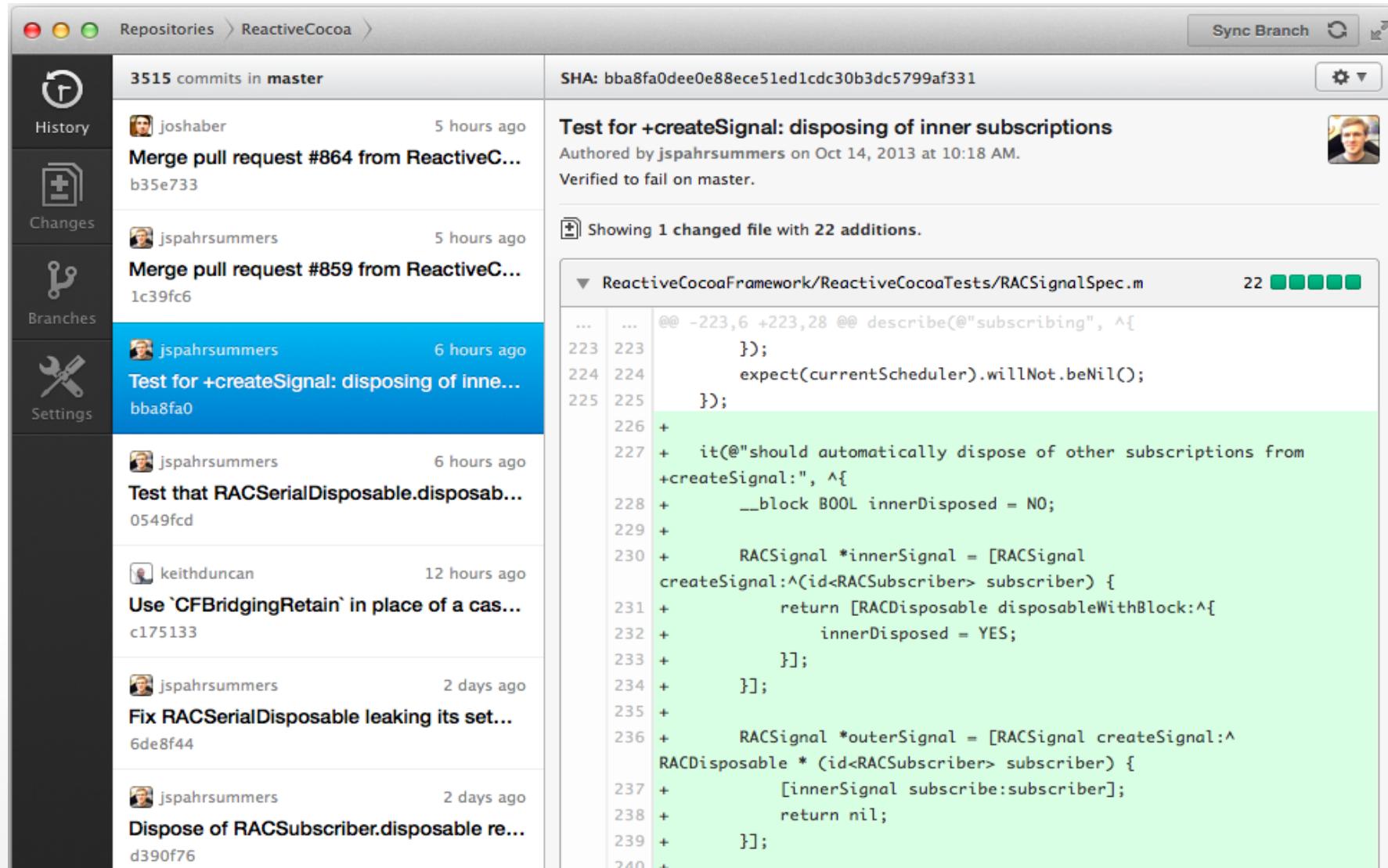
## ReactiveUI

Use the Reactive Extensions for .NET along with Silverlight, WPF, or Windows Phone to create elegant, testable User Interfaces.

This library is organized into several high-level assemblies:

- **ReactiveUI** - Core library that doesn't rely on any particular UI framework. `ReactiveObject`, the base `ViewModel` object, as well as `ReactiveCollection`, a more awesome `ObservableCollection`, is in here.
- **ReactiveUI.Xaml** - Classes that require references to a Xaml'y framework, like WPF or WinRT. `ReactiveCommand`, an implementation of `ICommand`, as well as the `UserError` classes are in this assembly.
- **ReactiveUI.Blend** - This class has several Blend Behaviors and Triggers that make attaching `ViewModel` changes to Visual State Manager states.
- **ReactiveUI.Routing** - A screens and navigation framework as well as `ViewModel` locator. This framework helps you to write applications using IoC containers to locate views, as well as navigating back and forwards between views.

# 5. Rôzne GUI pre git -MacOS -GitHub



# 5. Rôzne GUI pre git -Linux -qgit

File Edit View Actions Help

Short log | 82f3906bd9bc733711725ed40f8b51fd484f6904

Git tree

Rev list

Graph | Short Log | Author | Author Date

Working directory changes  
master fixed the test for 'make all parameters optional' in UserSet  
origin/master 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...

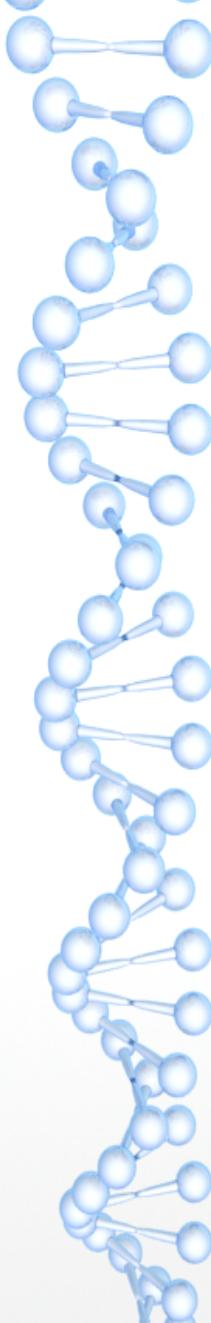
core/admin.py  
core/templates/admin/core/systemtext/change\_

```
return response.TemplateResponse(request, 'admin/core/systemtext/push_r
    {'notification': notification, 'num_devices': num_devices, 'sent': r
        current_app='core')

@@ -272,6 +297,11 @@ @admin.site.register_view('cli', 'Command line interface')
admin.site.register_view('dbDumps', 'Db manipulation', view=DbDumps.as_view)
admin.site.register_view('gitTags', 'Git versions of this site', view=GitT

+class GetLanguages(View):
+    """Helper page for getting all the languages in JSON"""
+    def get(self, request, *args, **kwargs):
+        return JsonResponse(json.dumps([models.model_to_dict(L) for L in mode
+@admin.site.register_view('getLanguages', 'Helper JSON getter for all Languages')
```

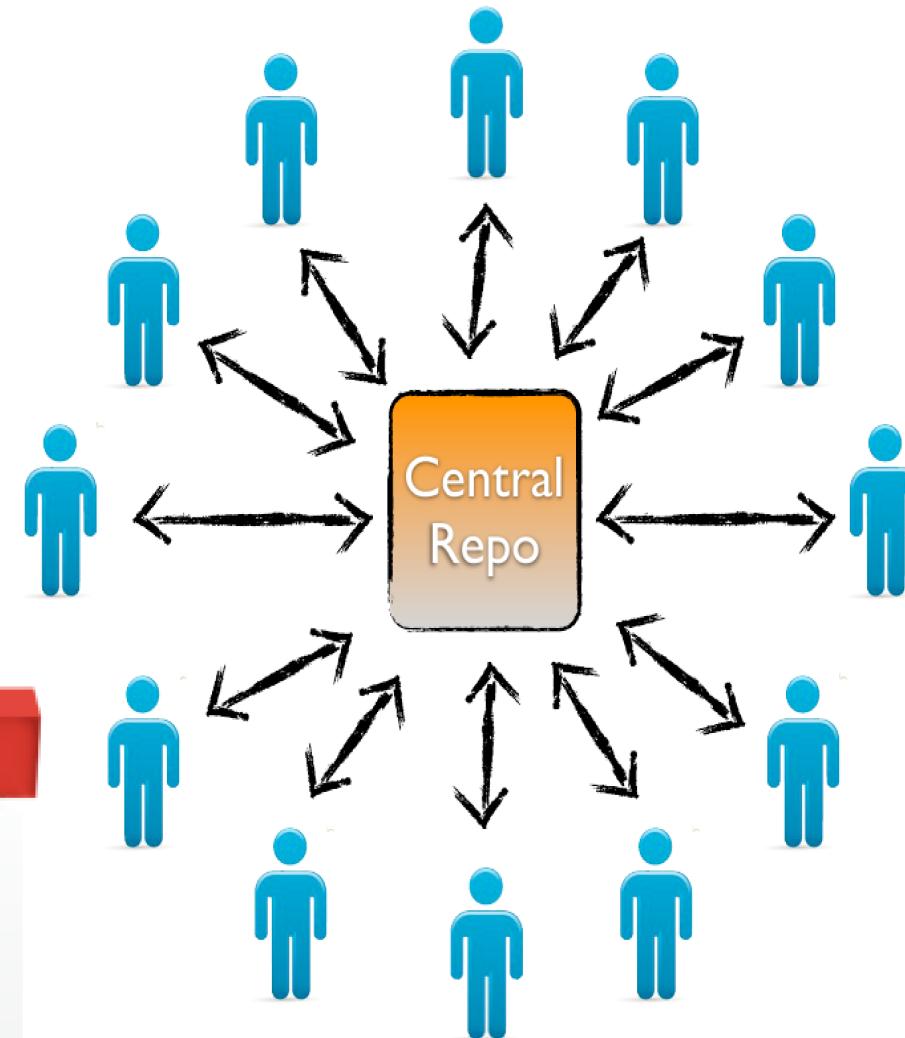
Tag: v0.122 [Feature #2209]



## 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

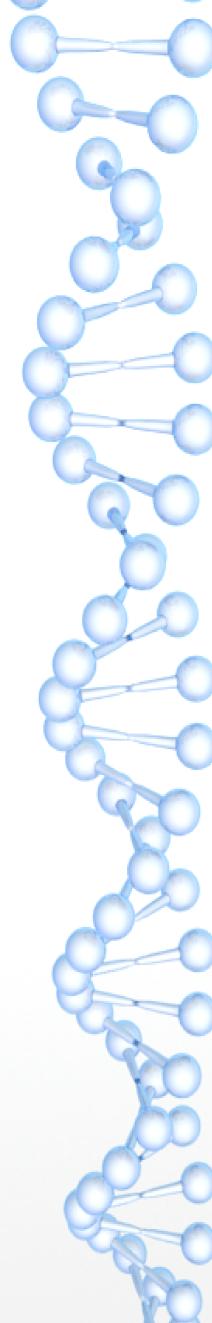
# 6. „Tvar“ dev komunity -centrálny renozitár



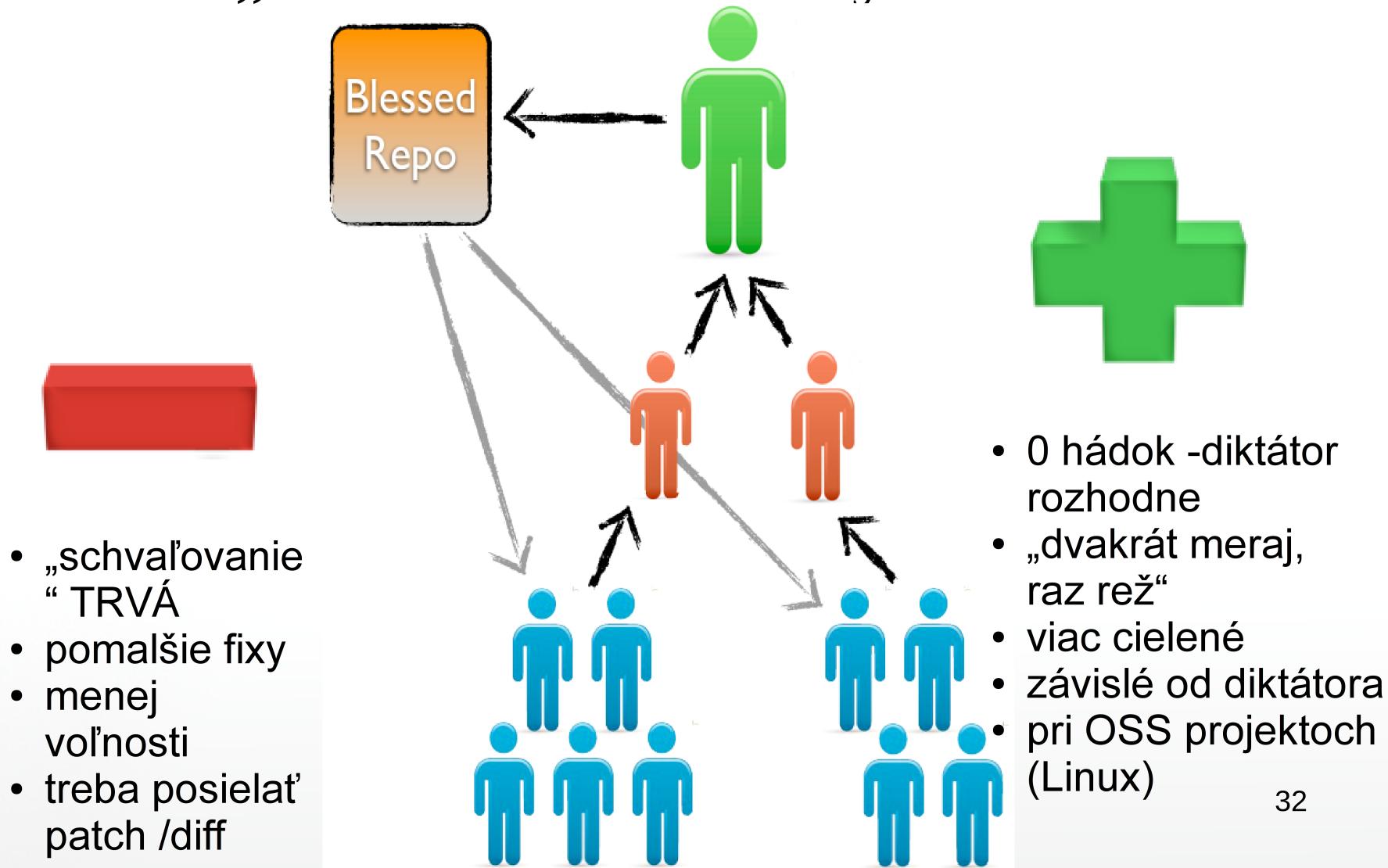
- stabilita buildu
- kolízie závýmu

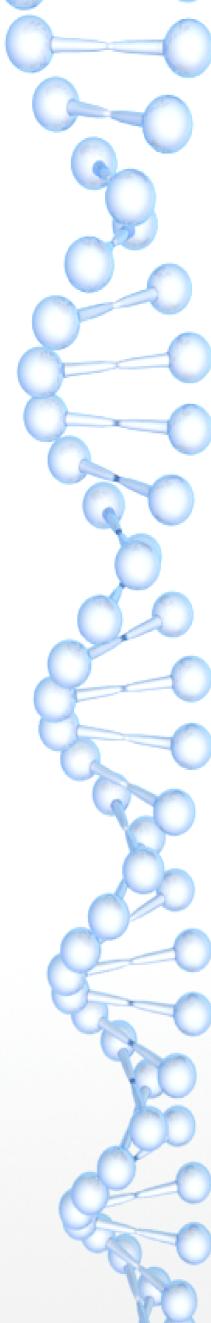


- rýchlosť
- jednoduchosť
- v TB internet banking

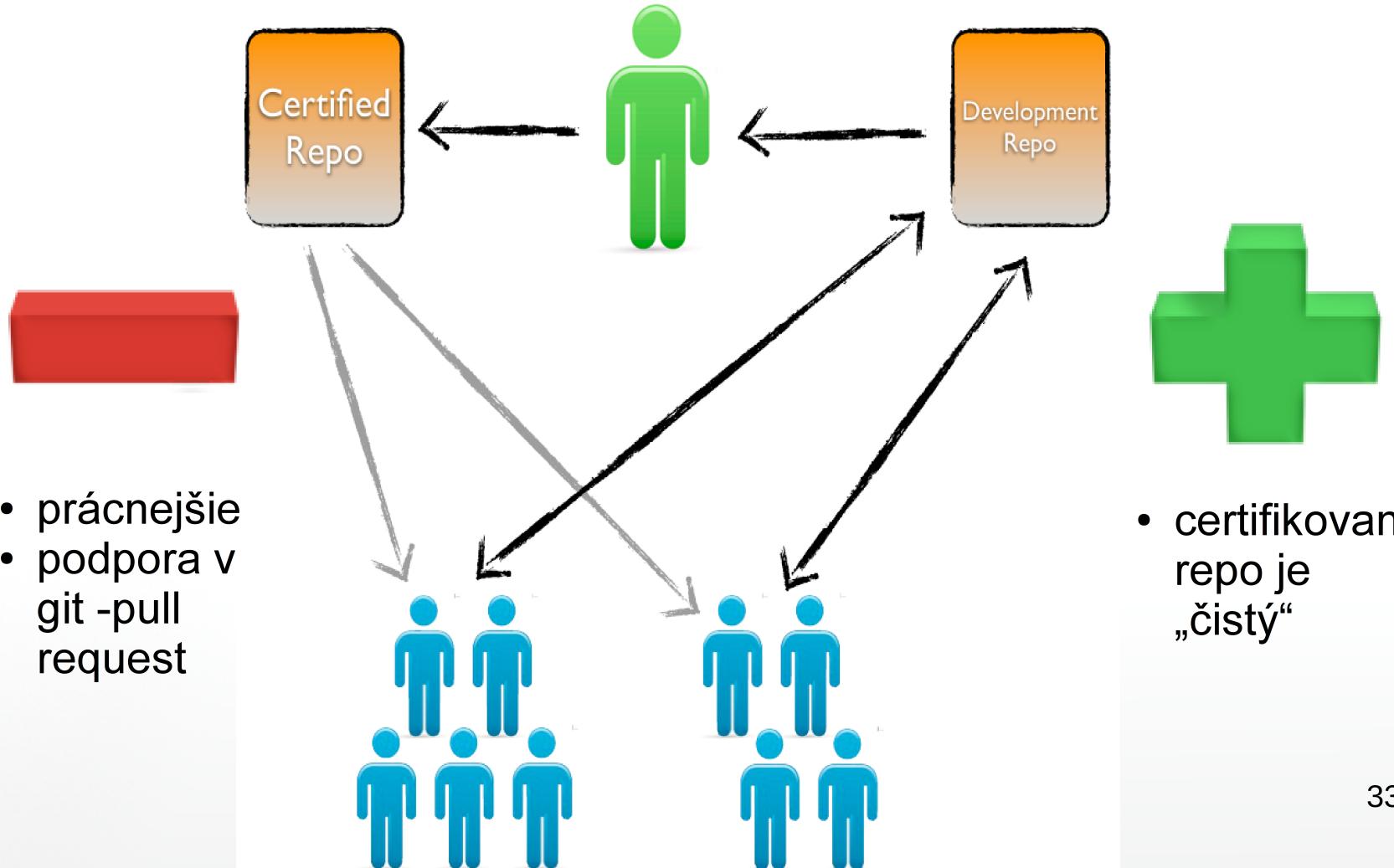


# 6. „Tvar“ dev komunity -diktátor

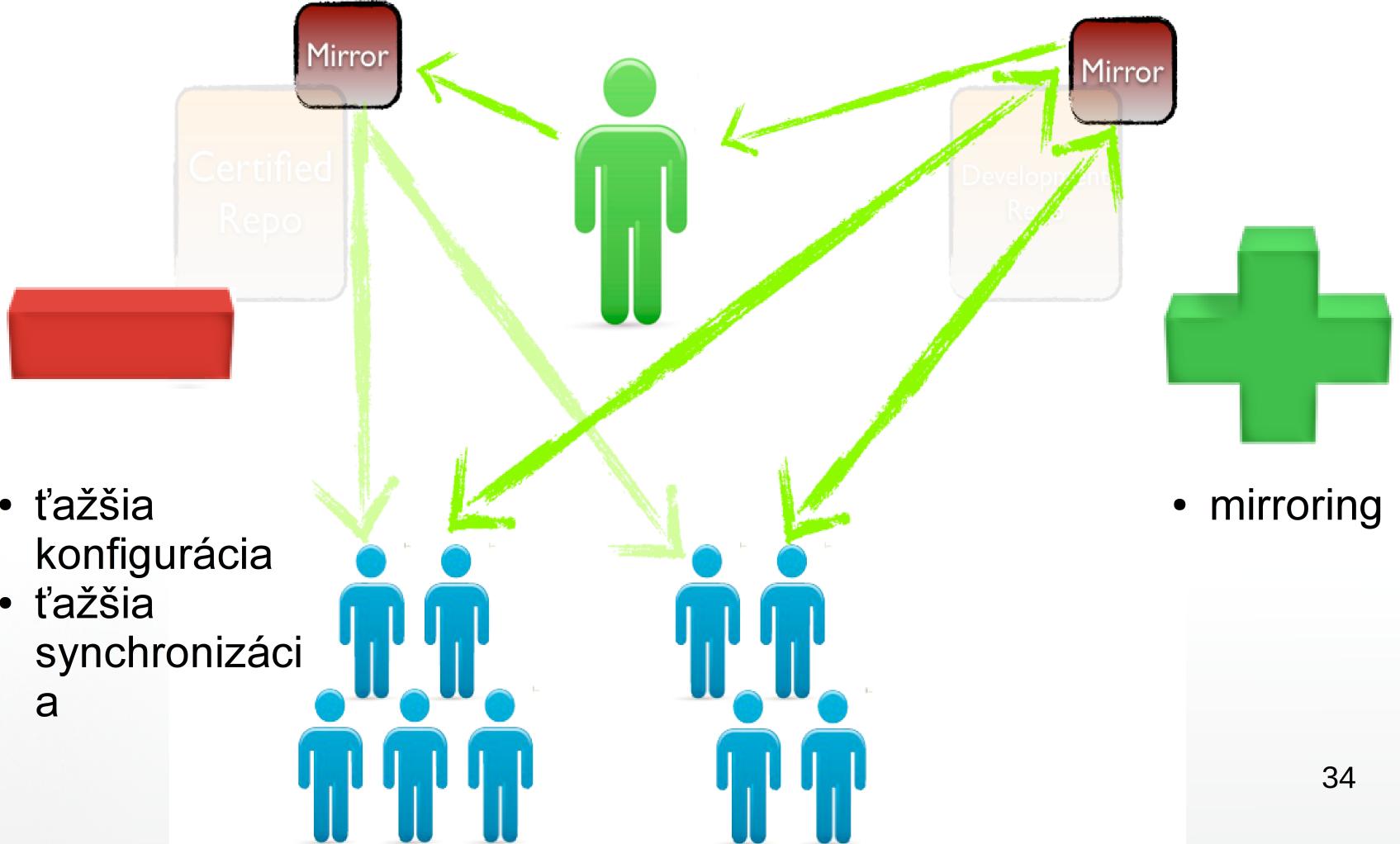




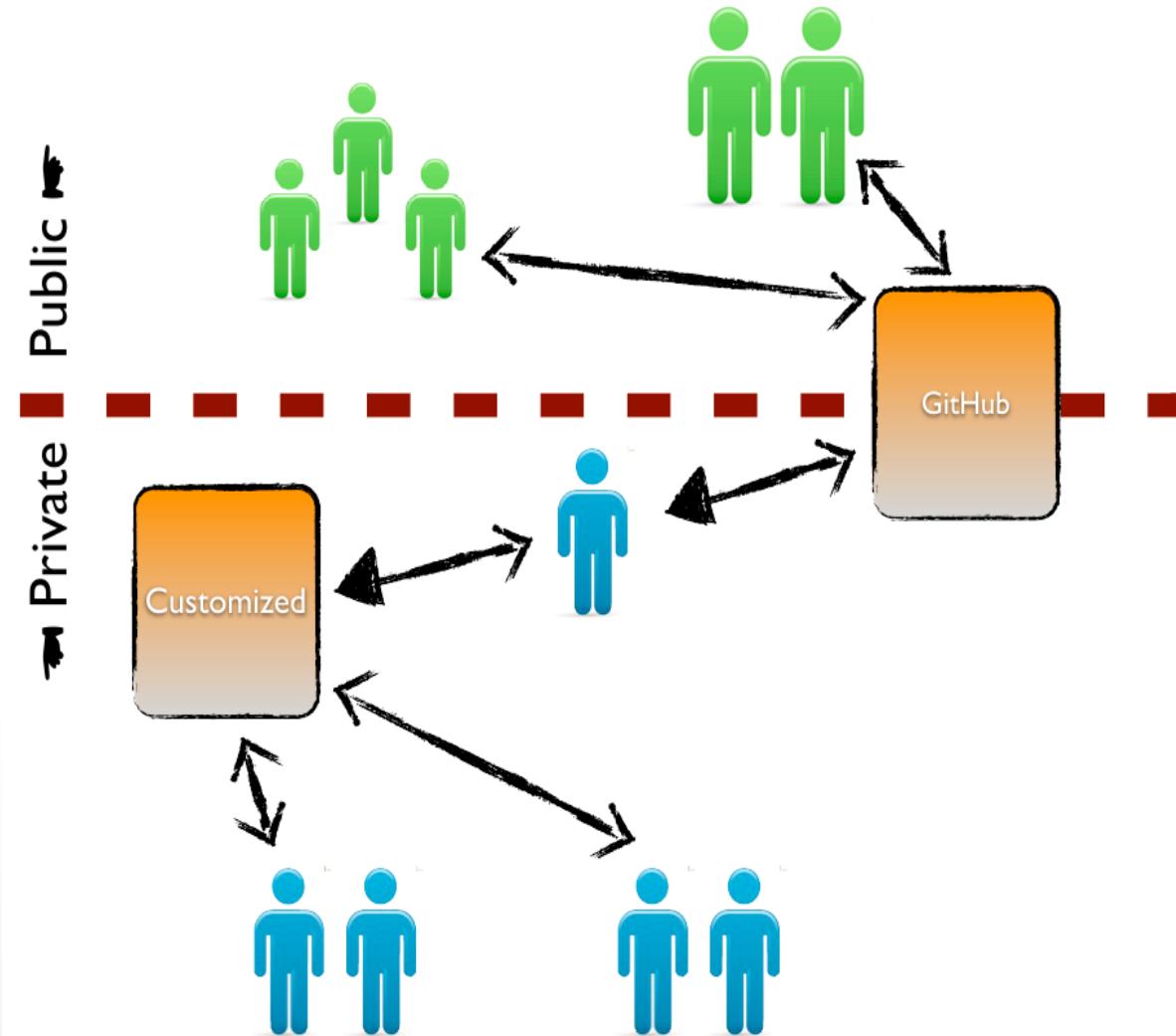
## 6. „Tvar“ dev komunity -diktátor v2

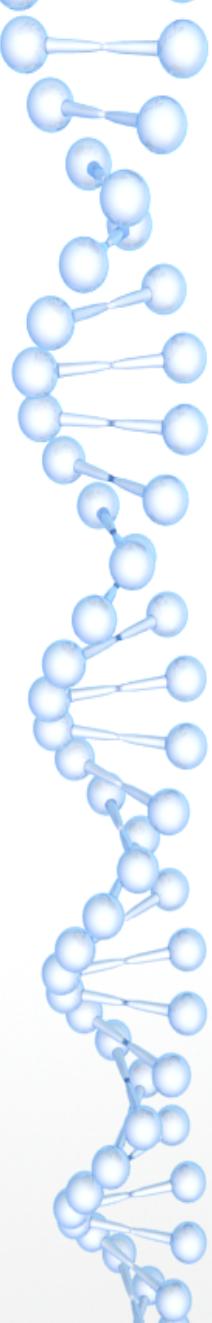


# 6. „Tvar“ dev komunity -diktátor v2 +mirror



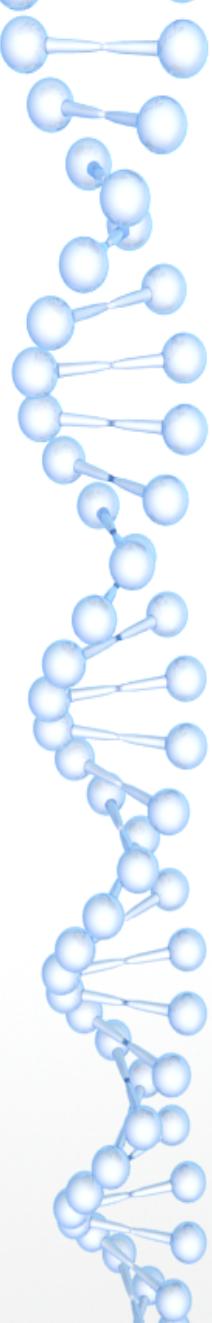
## 6. „Tvar“ dev komunity -open +closed





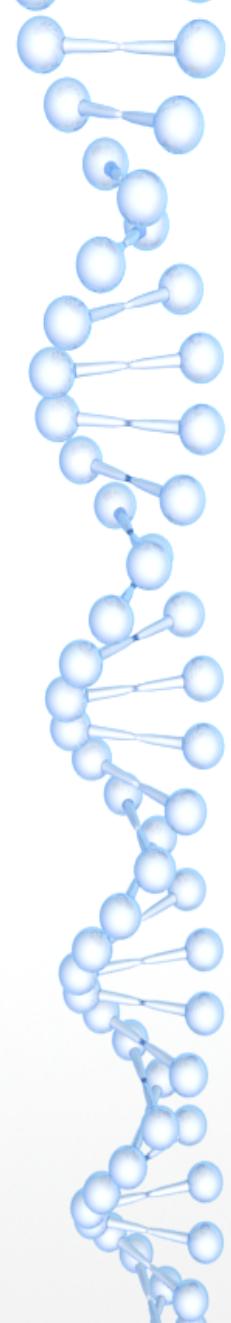
## 6. Sumár (alebo „Čo už vieme“)

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



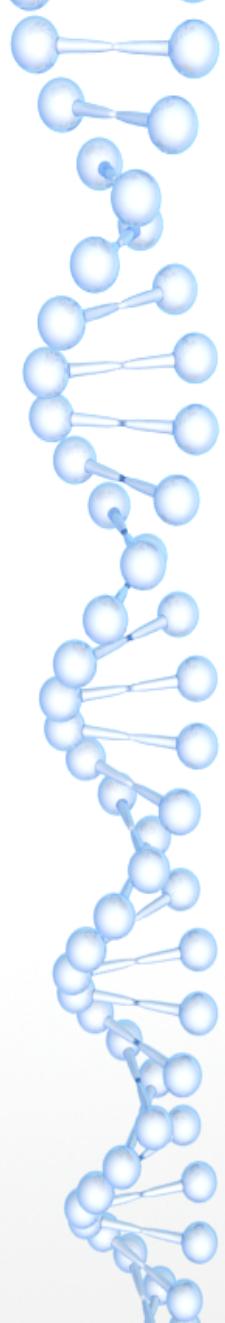
## 7. Záverom

- 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 *automatizovanú* prácu -nemožno od neho čakať „telepatiu“ (viď Merge-ovanie)
- GUI nástroje sú fajn, ale vždy je lepšie vidieť im na prsty



## Otázky?

(pokojne ich dajte aj do súkromnej  
komunikácie na Bitrix-e projektu  
MojaKomunita)



Ďakujem za pozornosť!