Git workshop

Aj Ty v IT

Tibor Stanko
18.10.22

1 Úvod

O mne

- **Tibor Stanko**, 31 rokov
- od 2020 dátový inžinier v Zurich Insurance, Bratislava SK
- predtým 6 rokov v akademickej sfére vo FR (PhD, postdoc)
- rád automatizujem nudné úlohy s pomocou Pythonu 🐍
- nie som Git guru, no Git používam denne už viac ako 7 rokov
- moje voľnočasové aktivity: 👨‍👧 🚵 ⛰️ 🎸 🎹 🍺

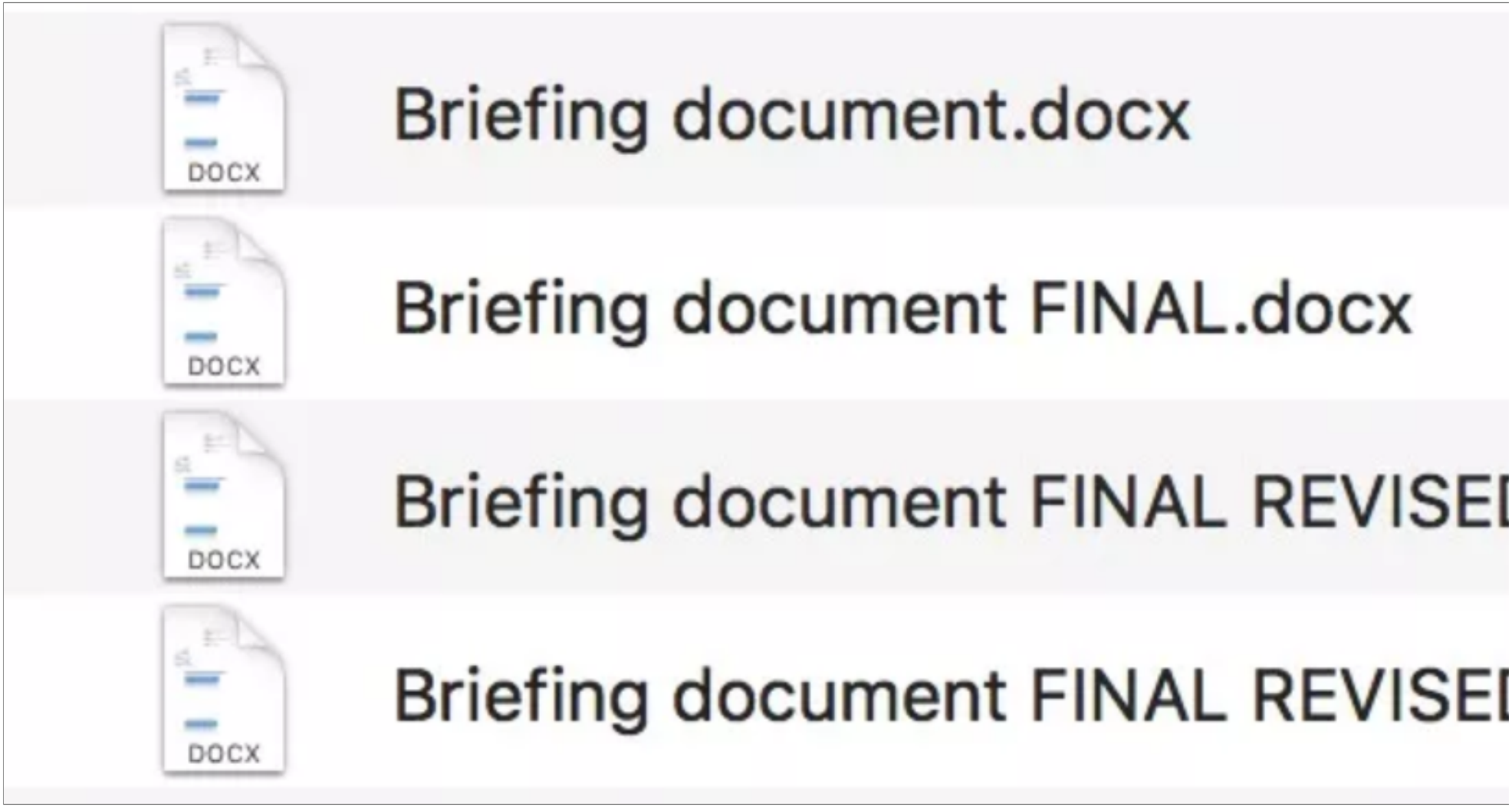
Obsah tohto workshopu

2 Git a Github

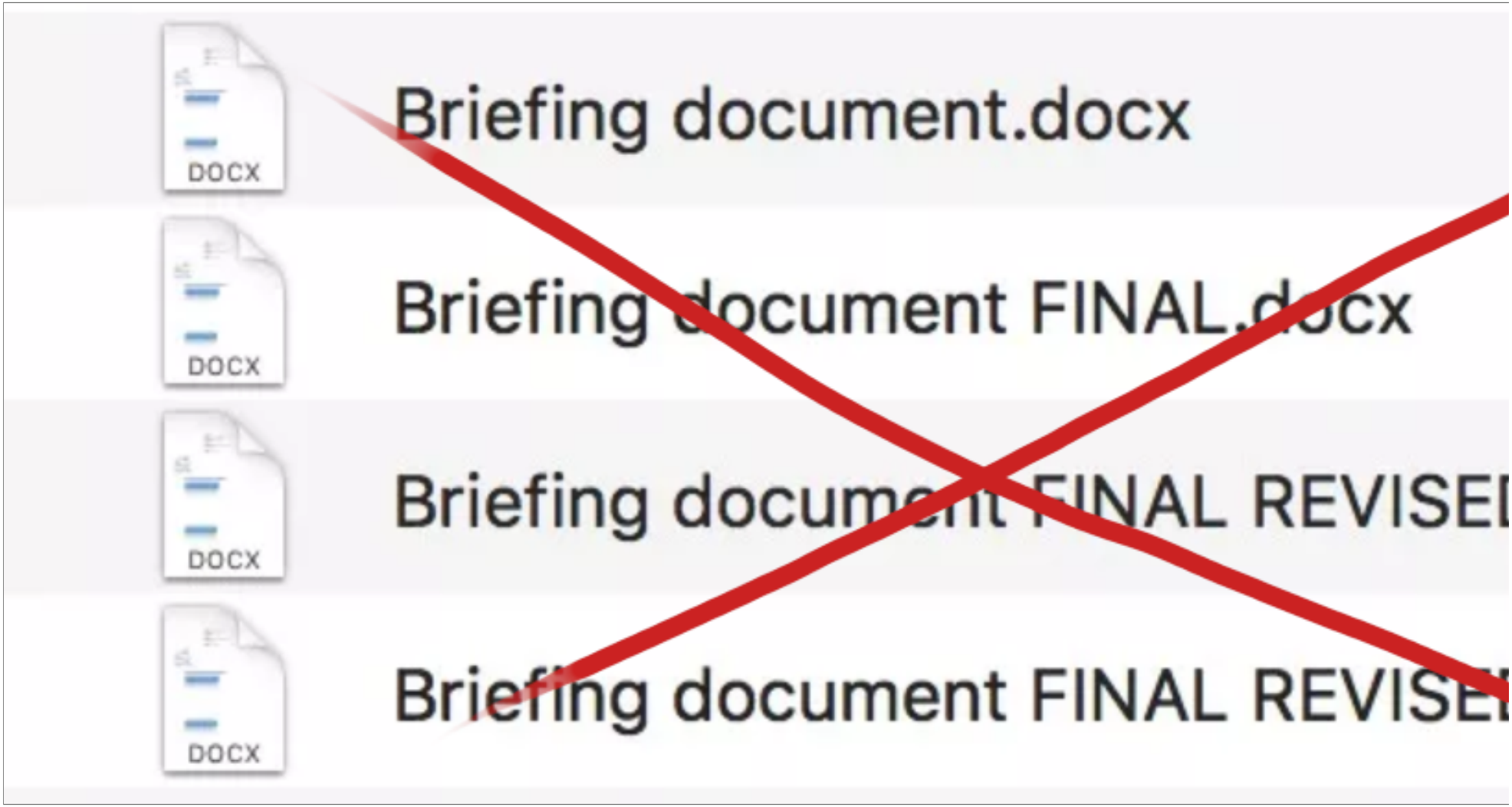
Čo je to Git?

- **systém riadenia verzií**
- angl. *version control system* (VCS) alebo *source control management* (SCM)
- zaznamenáva históriu vývoja projektu
- užitočný pre tímy aj pre jednotlivcov
- nie je len o kóde, dovoľuje ukladať ľubovoľné súbory (aj netextové)

Prečo potrebujeme systémy riadenia verzií?



Prečo potrebujeme systémy riadenia verzií?



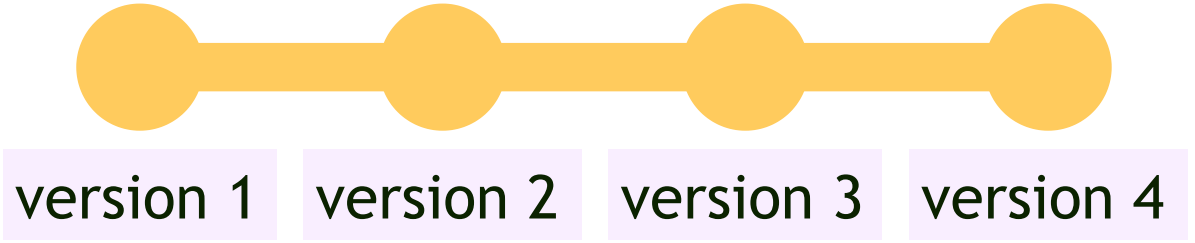
Čo umožňuje Git?

- **ukladať** verzie
- **prepínať** medzi verziami
- **obnoviť** predošlú verziu
- **porovnávať** verzie
- vytvárať **vetvy**
- **zlučovať** vetvy
- **zálohovať** súbory

Čo je to Github?

- *"sociálna sieť pre programátorov"*
- **Git** = systém riadenia verzií
- **Github** = cloudový portál na ukladanie repozitárov
- používa Git, no nie je jeho súčasťou
- obsahuje funkcionalitu ktorá nie je v Gite
  - *Issues*, *Pull requests*, *Actions*, …
- podobné služby: **Gitlab**, **Bitbucket**, **Azure DevOps**, a iné

3 Základy Gitu



Krok 0: Pracujeme s terminálom

- Aby sme pochopili ako Git funguje, budeme na začiatku spúšťať Git cez **terminál**
- Neskôr si ukážeme aj použitie Gitu priamo v IDE (napr. vo VS Code)
- Vo **Windowse** odporúčam **Windows Terminal** s **PowerShell** (built-in) alebo **Nushell**
- Základné príkazy na navigáciu medzi adresármi v termináli:
  - `pwd` — vypíš aktuálny adresár
  - `cd folder` — zmeň aktuálny adresár na `folder`

```
>> pwd                          # C:/Users/tibor.stanko
>> cd folder                    # C:/Users/tibor.stanko/folder
>> cd ..                        # C:/Users/tibor.stanko
>> cd C:/Users/janko.hrasko     # C:/Users/janko.hrasko
>> cd ~                         # C:/Users/tibor.stanko
```

Krok 1: Inštalácia Gitu

- Vo **Windowse** existuje viacero spôsobov ako nainštalovať Git, napr.
  - cez **klasický inštalátor**
  - cez **scoop** (odporúčam) — v PowerShell spustite nasledovné príkazy:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
irm get.scoop.sh | iex # inštalácia scoopu
scoop install git # inštalácia gitu
```

- Inštrukcie pre **macOS** alebo **Linux** sú na oficiálnej stránke **v angličtine** aj **v češtine**.

Krok 2: Nastavenie Gitu

- Skontrolujme či je Git správne nainštalovaný:

```
git --version
```

- Predtým ako začneme pracovať s Gitom, je potrebné nastaviť si meno a email cez príkaz `git config` . Tieto údaje bude Git používať na priradenie autora k verzii.

```
git config --global user.name "Tibor Stanko"
git config --global user.email "tibor.stanko@zurich.com"
```

- Predvolenú vetvu si nastavíme na `main` (o **vetvách** si povieme viac neskôr)

```
git config --global init.defaultBranch "main"
```

Krok 3: Vytvorenie Git repozitára

- Prepneme sa do adresára z ktorého chceme spraviť repozitár:

```
cd ~/hello
```

- Príkaz `git init` slúži na vytvorenie Git repozitára v aktuálnom adresári:

```
git init
```

- Po spustení príkazu Git vypíše:

```
Initialized empty Git repository in C:/Users/tibor.stanko/hel
```

Krok 3: Vytvorenie Git repozitára

- Stav repozitára môžeme skontrolovať cez `git status` :

```
git status
```

- Takto vyzerá stav prázdneho Git repozitáru (žiadne súbory ani uložené verzie):

```
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to tra
```

Krok 4: Uloženie verzie

Uloženie verzie prebieha v dvoch krokoch.

1. Cez `git add` označíme zmeny ktoré majú byť pridané do novej verzie:

```
git add hello.py
```

2. Cez `git commit` vytvoríme záznam o novej verzii:

```
git commit -m "Add hello.py"
```

Krok 4: Uloženie verzie

- Popis záznamu *(commit message)* je väčšinou krátka jednoriadková správa ktorá sa špecifikuje cez argument `-m "commit message"`
- Ak chceme napísať dlhší popis, argument `-m` vynecháme. `git commit` vtedy otvorí textový editor v ktorom popis napíšeme.
- V Gite na Windowse je na písanie *commit message* predvolený editor `vim`, ktorý beží priamo v termináli. Ak nemáte skúsenosti s používaním vimu, môžete si editor zmeniť napr. na `notepad`:

```
git config --global core.editor notepad
```

Krok 4: Uloženie verzie

Neflákajte písanie *commit messages*!

Prečo verziu ukladáme v dvoch krokoch?

Niektoré systémy na kontrolu verzií fungujú tak že vytvoria novú verziu zo všetkých aktuálnych súborov v repozitári. Tento spôsob ukladania záloh môže byť nevýhodný. Príkladom je situácia keď sme v repozitári implementovali dve nezávislé funkcie, a chceme ich zachytiť v dvoch rozdielnych verziách. V Gite preto existuje koncept prípravnej zóny (*staging area*), vďaka ktorej máme kontrolu nad tým ktoré zmeny budú a ktoré nebudú pridané do nasledujúcej verzie.

```
# 1. add/stage - pridaj súbor prípravnej zóny
git add test.txt
# 2. commit - ulož novú verziu
git commit -m "added test.txt"
```

Krok 5: Kontrola stavu repozitára

- Aktuálny stav repozitára môžeme skontrolovať cez `git status` :

```
On branch main
nothing to commit, working tree clean
```

- Cez príkaz `git log` sa môžeme presvedčiť o tom že záznam *(commit)* bol vytvorený:

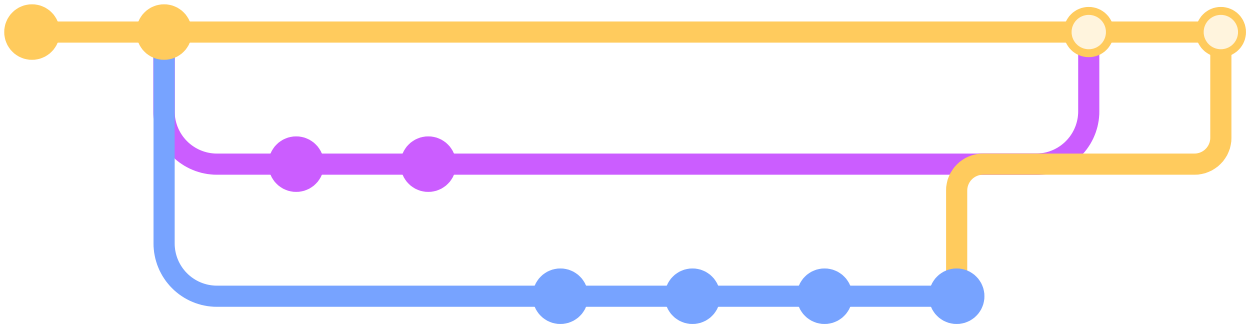
```
commit bf5c9b4a320012b422546fcb86f5b957104bea55 (HEAD -> mair
Author: Tibor Stanko <tibor.stanko@gmail.com>
Date:   Tue Sep 13 17:00:00 2022 +0200

    Add hello.py
```

Úlohy (1)

1. Vytvor si na svojom počítači prázdny adresár `zoo`
2. Sprav z adresára `zoo` Git repozitár
3. Vytvor v repozitári súbor `test.txt` s ľubovoľným obsahom
4. Pridaj `test.txt` do ďalšej verzie a ulož ju
5. Skontroluj novú verziu cez `git log`

Tip: nezabudni počas práce používať `git status` na zistenie aktuálneho stavu repozitára.

4 Vetvy *(branches)*



Čo je to vetva *(branch)* ?

- Vetvy slúžia na vybočenie z hlavnej línie a pokračovanie v práci bez zásahu do nej
- Vetvenie je silnou stránkou Gitu — prepínanie medzi vetvami je rýchle, čo umožňuje časté vytváranie nových vetiev
- Doteraz sme pracovali na vetve `main` , ktorú automaticky vytvoril `git init`
- Zoznam vetiev si môžeme pozrieť cez príkaz `git branch` :

```
>> git branch
* main
```

V Gite je prednastavená hlavná vetva `master` . Tento názov sa v posledných rokoch **stal kontroverzným** a postupne sa **prestáva používať**. Preto sme pri **nastavovaní Gitu** zmenili `init.defaultBranch` na `main` .

Na čo sú dobré vetvy?

- Vetvy umožňujú efektívne pracovať paralelne na viacerých častiach projektu
- Príkladom je situácia keď vytvárame novú funkcionalitu (vetva A), a vyskytne sa bug ktorý musí byť hneď opravený (vetva B)
- Vďaka Gitu môže práca na týchto dvoch vetvách prebiehať nezávisle

Vytvorenie vetvy *(branch)*

- Na vytvorenie novej vetvy s názvom `french` zavoláme

```
git branch french
```

- Ak sa chceme prepnúť na novú vetvu:

```
git checkout french
```

- Tieto dve operácie sa veľmi často robia spolu, preto existuje skratka:

```
git checkout -b slovak # vytvor vetvu 'slovak' a prepni sa na
```

Zlučovanie vetiev *(merge)*

- Ak chceme pridať zmeny spravené na vetve `slovak` do hlavnej vetvy `main`, prepneme sa najprv na hlavnú vetvu:

```
git checkout main
```

- Na zlučovanie vetiev použijeme príkaz `git merge`:

```
git merge slovak
```

Zlučovanie vetiev *(merge)*

Auto-merge

- Ak je to možné, Git automaticky zlúči zmeny z oboch vetiev. V takom prípade uvidíme nasledovný výstup:

```
>> git merge slovak
Auto-merging hello.py
Merge made by the 'ort' strategy.
 hello.py | 12 ++++++++++-
 1 file changed, 11 insertions(+), 1 deletion(-)
```

Zlučovanie vetiev *(merge)*

Fast-forward

- V prípade že zlučovaná vetva je priamym potomkom cieľovej vetvy, Git urobí tzv. *fast-forward*:

```
>> git merge comment
Updating 3a5d22e..9ad633c
Fast-forward
 hello.py | 1 +
 1 file changed, 1 insertion(+)
```

Zlučovanie vetiev *(merge)*

Riešenie konfliktov

- Ak automatické zlúčenie zlyhá, Git nám vo výstupe nahlási *merge conflict*. V takomto prípade musia byť zmeny z oboch vetiev zlúčené manuálne.

```
>> git merge french
Auto-merging hello.py
CONFLICT (content): Merge conflict in hello.py
Automatic merge failed; fix conflicts and then commit the res
```

- Po manuálnom zlúčení je potrebné pridať zmenené súbory do novej verzie:

```
git add hello.py
git commit -m "Merged branch 'french'"
```

Vymazanie vetvy *(delete)*

- Ak po zlúčení vetvu už nepotrebujeme, môžeme ju vymazať cez `git branch --delete` , skrátene `git branch -d` .

```
git branch -d comment
```

⚠ *Po vymazaní je vetva odstránená z histórie a nie je možné ju obnoviť.*

Príklad `git log` histórie

```
*   3a5d22e (HEAD -> main) Merge branch 'french'
|\
| * 21c7ab7 (french) Add french functionality
* |   1364948 Merge branch 'slovak'
|\ \
| * | c3159a6 (slovak) Add slovak functionality
| |/
* / 67e86d0 Fix missing exclamation mark
|/
* de1543b Add hello.py
```

Úlohy (2)

1. Vo svojom lokálnom repozitári sa prepni na novú vetvu `animals`
2. Vytvor v repozitári nový súbor `zoo.txt` s nasledovným obsahom:

```
panda
slon
lev
zirafa
```

3. Ulož novú verziu ktorá bude obsahovať `zoo.txt`
4. Zlúč zmeny spravené na vetve `animals` do vetvy `main` a vymaž vetvu `animals`
5. Na novej vetve `tiger` zmeň riadok `lev` na `tiger` a ulož novú verziu
6. Prepni sa na vetvu `main`, oprav riadok `zirafa` na `žirafa` a ulož novú verziu
7. Zlúč zmeny z vetvy `tiger` s vetvou `main`

5 Vzdialené repozitáre *(remotes)*

Čo je to vzdialený repozitár *(remote)?*

- Doteraz sme pracovali s **lokálnym** Git repozitárom ktorý je uložený na našom počítači
- **Vzdialený repozitár** *(remote)* je uložený na Internete — presnejšie, na webovom serveri
  napr. github.com, firemný server, univerzitný server, …
- Existujú dva typy *remote* repozitárov:
  a. **verejný** *(public)* repozitár je zdieľaný so všetkými používateľmi s prístupom na server
  b. **súkromný** *(private)* repozitár je zdieľaný iba s vybranými používateľmi

Na čo slúžia vzdialené repozitáre?

1. **Zálohovanie** kódu
2. **Zdieľanie** kódu
3. **Synchronizácia** kódu v tíme

Vytvorenie repozitára na Githube — github.com/new

Vytvorenie repozitára na Githube — github.com/new

Vytvorenie repozitára na Githube — github.com/new

Vytvorenie repozitára na Githube — github.com/new

Vytvorenie repozitára na Githube — github.com/new

Nastavenie *remote* v lokálnom repozitári

- Na nastavenie vzdialeného repozitára použijeme príkaz `git remote add <name> <url>` :

```
git remote add origin https://github.com/bbrrck/zoo.git
```

- `name` bude Git používať ako meno vzdialeného repozitára na adrese `url` . Meno môže byť ľubovoľné; bežne sa stretneme s menom `origin` .
- Lokálny repozitár môže mať priradený aj viac ako jeden *remote*.

Poslanie lokálnej kópie na *remote*

- Príkaz `git push <remote> <branch>` "pretlačí" lokálne zmeny z vetvy `branch` do vzdialeného repozitára `remote` :

```
git push origin main
```

- Pri prvom zavolaní `git push` je potrebné pridať argument `-u` :

```
git push -u origin main
```

- `-u` alebo `--set-upstream` nastaví predvolenú *remote* vetvu (`origin/main`) pre aktuálnu lokálnu vetvu (`main`)
- ak *remote* vetva `origin/main` neexistuje, `git push` ju automaticky vytvorí

Príklad výstupu z `git push`

```
git push -u origin main
```

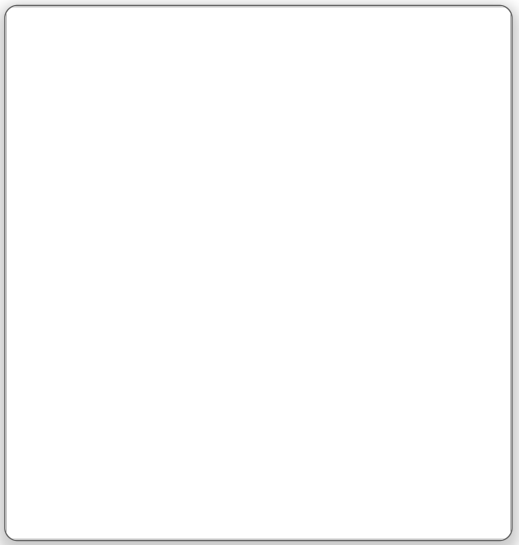
- Výstup:

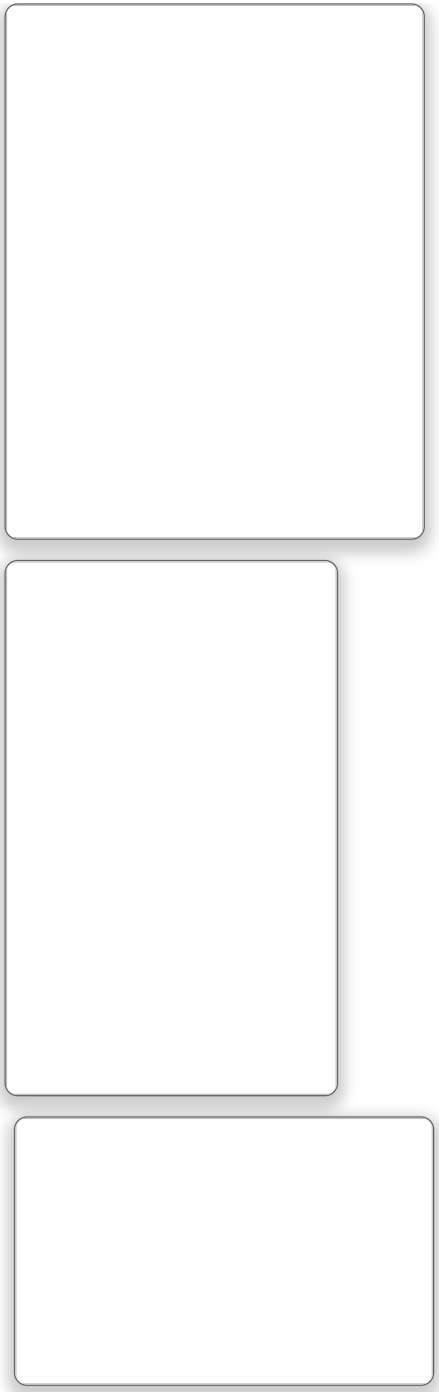
```
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 241 bytes | 120.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/bbrrck/test-repo-01.git
 * [new branch]      main -> main
```

```
branch 'main' set up to track 'origin/main'.
```

Odbočka: prihlásenie do Githubu

- Aby mohol Git posielať dáta na Github, je potrebná autentifikácia
- Najjednoduchší spôsob na správu prihlasovacích udajov: **Git Credential Manager**
  (je súčasťou **Git for Windows**)
- Konfigurácia sa začne automaticky po prvom spustení `git push`





Zoznam vetiev

- `git branch -a` vypíše zoznam všetkých vetiev, lokálnych aj vzdialených
- `-a` je skratka pre `--all`

```
git branch -a
```

- Výstup — `*` označuje aktuálnu vetvu:

```
* main
  tiger
  remotes/origin/main
```

Konflikt: *remote* zmeny neexistujú lokálne

```
To https://github.com/bbrrck/zoo.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/bbrrck
hint: Updates were rejected because the remote contains work
hint: not have locally. This is usually caused by another rep
hint: to the same ref. You may want to first integrate the re
hint: (e.g., 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help'
```

Stiahnutie *remote* zmien do lokálnej vetvy

- Pomocou príkazu `git fetch` stiahneme zoznam zmien z *remote* vetvy:

```
git fetch
```

- Zmeny ktoré nie sú súčasťou lokálnej vetvy zlúčime pomocou `git merge` :

```
git merge # bez argumentov
```

- Vo väčšine prípadov môžeme jednoducho zavolať príkaz `git pull` , ktorý je kombináciou `git fetch` a `git merge` :

```
git pull
```

Klonovanie existujúceho *remote* repozitára

- Existujúci vzdialený repozitár môžeme naklonovať pomocou `git clone` :

```
git clone <remote_url> <local_folder>
```

- Tento príkaz vytvorí kópiu repozitára z `remote_url` v adresári `local_folder`
- Príklad:

```
git clone https://github.com/bbrrck/zoo.git myzoo
# alebo
git clone https://github.com/bbrrck/zoo.git # naklonuje sa do
```

Úlohy (3)

1. Vytvor si na Githube repo s názvom `zoo`
2. Pridaj tento *remote* do svojho lokálneho repa
3. Nahraj lokálnu vetvu `main` na *remote*
4. Na Githube pridaj do súboru `zoo.txt` za meno každého zvieraťa jeho emoji: 🐼🐘🐯🦌
5. Zosynchronizuj lokálne repo s Githubom
6. Na Githube pridaj do súboru `zoo.txt` nový riadok `krokodíl` 🐊 a ulož novú verziu
7. V lokálnom repe pridaj do súboru `zoo.txt` nový riadok `gorila` 🦍 a ulož novú verziu
8. Zosynchronizuj lokálne repo s Githubom a vyrieš vzniknutý *merge conflict*

6 Pokročilý Git

`.gitignore`

- špecifikuje ktoré súbory má Git ignorovať
- **.gitignore generátor**

`git <command> --help`

```
# vypíš pomoc (dokumentáciu) ku danému príkazu -- napr. git c
git commit --help
# alebo
git commit -h
```

`git diff`

```
# zobraz zmeny medzi dvoma verziami
git diff
git diff zoo.txt
git diff 235a0d8 5d94512 zoo.txt
git diff HEAD~1 HEAD zoo.txt
```

`git cat-file -p`

```
# zobraz obsah Git objektu (commit, tree, blob)
git cat-file -p 235a0d8
```

`git restore`

```
# pridaj súbor do prípravnej zóny (staging area)
git add zoo.txt
# odstráň súbor z prípravnej zóny (unstage)
git restore --staged zoo.txt
```

`git add --interactive`

```
# interaktívne pridaj zmeny
git add --interactive
# alebo
git add -i
```

`git log`

```
# prispôb výstup z git logu
git log --all --oneline --graph --decorate
# alebo
git config --global alias.nicelog "log --all --graph --decora
git nicelog
```

`git stash`

```
# dočasne odstráň zmeny vo working directory
git stash
# obnov odstránené zmeny
git stash pop
```

`git blame`

```
# zisti kto spravil poslednú zmenu v každom riadku
git blame
```

`git revert`

```
# odstráň zmeny vykonané v poslednom commite
git revert HEAD
```

Príkaz `git revert` vytvorí novú verziu, a nemení históriu repozitára.

`git reset`

```
# vráť repozitár do stavu po commite s commit_id
git reset --hard <commit_id>
```

⚠ *Príkaz* `git reset` *mení históriu repozitára a môže spôsobiť stratu súborov.*

7 Markdown rýchlokurz

Čo je to Markdown?

- **Markdown** je jednoduchý značkovací jazyk (ako HTML alebo TeX) ktorý sa používa na tvorbu rôznych typov obsahu: dokumenty, články, slidy, webstránky, …
- Markdown je *de facto* štandard pre dokumentovanie Git projektov
- Väčšina Github projektov má `README.md` ktorý Github automaticky vyrenderuje
- Príklady dobre napísaných `README` súborov: **matiassingers/awesome-readme**

*Aj tieto slidy boli vytvorené s použitím Markdownu! (pomocou systému **Quarto**)*

```
# Markdown is Awesome

Markdown is very simple and
versatile.

This is a Markdown paragraph.
This is still the same paragraph.

## Formmatting options

Bulleted list:

- *italic*
- **bold**
- ***bold and italic***
- ~~strikethrough~~
- [link]
(https://www.markdownguide.org/)
- `code`

Numbered list:

1. first item
2. second item
3. last item
```

Markdown is Awesome

Markdown is very simple and versatile.
This is a Markdown paragraph. This is still the same paragraph.

Formatting options

Bulleted list:

- *italic*
- **bold**
- ***bold and italic***
- ~~strikethrough~~
- **link**
- `code`

Numbered list:

1. first item
2. second item
3. last item

````
### Code blocks

```python
def main():
    print("hello!")

if __name__ == "__main__":
    main()
```

### Images

![Queen Elizabeth II]
(https://upload.wikimedia.org/wikiped
ia/commons/thumb/a/aa/Queen_Elizabeth
_II_1959.jpg/399px-
Queen_Elizabeth_II_1959.jpg)

### Blockquotes

> It’s worth remembering that it is
often the small steps, not the giant
leaps, that bring about the most
lasting change.
````

Code blocks

```python
def main():
    print("hello!")

if __name__ == "__main__":
    main()
```

Images

Blockquotes

> It's worth remembering that it is often the small steps, not the giant leaps, that bring about the most lasting change.

8 Užitočné zdroje

Odkazy

sk

- videokurz od Yablka: **Git a Github od základov**
- videokurz na kanáli **Informatika s Mišom**
- predmet *Základy softvérového inžinierstva* na FEI TUKE
  - časť 2: **Systémy pre správu verzií**
  - časť 3: **Práca s vetvami v systéme Git**

Odkazy

en

- ***Pro Git***, voľne dostupná oficiálna kniha, k dispozícii čiastočne aj **v češtine**
- **git - the simple guide**
- **Git tutoriály** od Atlassianu
- Coursera: **Introduction to Git and GitHub**
- Missing Semester of CS Education, **Lecture 6: Version Control (git)**
- **Learn how Git works internally with simple diagrams**
- **Markdown Guide**
- **Learn Markdown in 5 minutes**

Git slovník

en	sk
branch	vetva
clone	naklonovanie repozitára
commit	záznam
commit message	popis záznamu
conflict	konflikt medzi verziami
conflict resolution	riešenie konfliktov
diff	rozdiel medzi verziami
merge	zlúčenie vetiev
pull	stiahnutie vzdialených zmien
push	odoslanie lokálnych zmien
repository	repozitár, úložisko
remote	vzdialený repozitár
snapshot	snímka
staging area	prípravná oblasť (tiež *index*)
status	stav repozitára
version	verzia

Ďakujem za pozornosť! 👏