Správa verzí v systému GIT Documentation

Vydání 1.0.0

Jáchym Čepický

20.09.2016

Obsah

Obsah

KAPITOLA 1

Úvod

Systém správy verzí (Version control) je důležitou částí správy programů, dokumentů webových stránek a dalších informací. Změny jsou ukládávny do bloků, tzv. *revizí* - jedna revize odpovídá stavu celého projektu v určitý čas. Systém správy verzí jde k počátkům knihtisku, kdy se jednotlivá vydání knihy označují datem nebo číslem vydání. Dalším úkolem systémů správy verzí je nějak umožnit spolupracovat více lidem na jednom souboru a "sehrávat" dohromady změny na různých ale i stejných místech (pomáhají tak řešit vzniklé *konflikty*).

Ke správě verzí se nejčastěji používají speciální programy. Často je správa verzí zapracována do softwarových produktů (MS Word) nebo služeb (Wikipedia).

Spravovat verze softwaru můžeme i "ručně" prostřednictvím vytváření kopií existujícího software. V praxi to ale vede ke spoustě chyb, málo kdo je tak disciplinovaný, aby pro každý pokus v konfiguračním souboru vytvořil nejprve stabilní kopii nebo naopak vytvořil kopii na hraní.

Občas je potřeba pracovat na několika verzích najednou, mít možnost mezi nimi rychle přepínat, vrátit se v historii, "odskočit si" do "verze z verze" a vyzkoušet jiný postup a tak dále.

1.1 Struktura použitá ve verzovacích systémech

1.1.1 Změny vs. celé soubory

Verzovací systémy pracují s daty v čase a ty lze uložit do různých struktur. Některé systémy ukládají pouze změny mezi soubory, což je výhodné při menších změnách ale komplikuje práci při komplexních změnách, jiné naopak ukládají celé souboury a zpětně dopočítávají změny mezi nimi, což je právě výhodnější při práci s komplexními změnami.

Aby mohla být změna v nějakém souboru zaznamenána v sytému, musí být nejprve *commitnuta*. *Necommitnutá* změna zatím nezaznamenaná v systému správy verzí je *pracovní kopie*. Je to podobné, jako když otevřete soubor, provádíte změny a *dokud změny nezapíšete zpátky na disk, změny se v souboru neprojeví*.

1.1.2 Centrální vs. distribuovaný

Způsob uspořádání pracovních kopií a různých verzí a větví může být buď *centrální*, kdy se všichni pracovníci vztahují k jednomu centrálnímu repozitáři nebo naopak *distribuovaný*, kdy každý uživatel je pánem své vlastní verze celého programu - žádná kopie není striktvně autoritativní.

1.1.3 Struktura revizí

I když je praxe často komplikovaná, nejbližší aproximací ukládání změn v systému verzí je představa stromu s centrálním hlavní kmenem, z něhož jsou odvozeny větve (které by se měly ideálně vracet zpět dohlaního kmene).

Každá změna zaznamenaná v kódu je označena jako *revize*. *Revize* je jedinečná sada změn v čase, označená unikátním identifikátorem.

Hlavní "větev", na které se účastníci projektu dohodli jako na větvi, kde se budou shromažďovat finální změny se označuje jako *trunk* nebo *master*.

Nová revize může vzniknout i sloučením předchozích větví - *mergováním*. Nová revize tak má za "rodiče" více než jednu větev.

1.2 Klíčová slova

Branch - Větev Sada souborů, jejichž kopie byla v určitém čase uložena do zvláštní větve a jsou na nich prováděny separátní změny. Takže obě kopie těchto souborů povedou k různému výsledku.

Diff - Výpis změn Většinou textový výpis znázorňující změnu mezi dvěma revizemi

Checkout - čekautovat Vytvořit lokální kopii revizí ze vzdáleného repozitáře.

Clone - klonovat Vytvořit nový repozitář obsahující revize z jiného repozitáře.

Commit - komit Zapsat nebo *mergovat* změny z neuložené *lokální kopie* zpět do repozitáře.

Conflict - konflikt Konflikt vznikne, pokud dvě strany provedly změny v souborech a systém je není schopen sám automaticky spojit. Uživatel obvykle musí konflikty vyřešit - *resolve* - ručně vybráním preferované verze nebo jejich ručním sloučením.

HEAD Revize, na kterou ukazuje aktuální verze - většinou poslední revize v dané větvi.

Merge - merdžovat Operace, během které jsou na jeden soubor aplikovány dvě sady změn.

Pull, push - pulovat, pušovat Kopírování revizí z jednoho repozitáře do druhého. Buď z cílového repozitáře "do mého" nebo kopírovat změny z mého lokálního repozitáře do cílového.

Repository - repozitář Místo, nejčastěji na serveru, ve kterém jsou kopie souborů spolu s jejich kompletní historií

Revize Změna v jakékoliv formě - je to stav celého vývojového stromu v nějakém čase. Jako synonymum se často používá *commit*.

Tag *Záložka*, speciálně pojmenovaný commit (revize) většinou odkazuje k nějaké významné události v čase projektu.

Trunk, master Unikátní vývojová linie která není větev (nebo naopak je "hlavní vývojová větev" záleží na pojmenování).

Working copy - pracovní kopie Změny sice uložené do souborů a zapsané na disk, ale zatím neuložené do repozitáře (pomocí *commitu*).

1.3 Na co je vhodné používat verzovací systémy

Nejvíce se hodí na textové soubory nebo malé binární soubory. Změny se totiž nejčastěji ukládají (a zobrazují) po řádcích. Binární soubory (jako jsou zazipované archivy, word dokumenty, velké obrázky a pod.) nemají více řádků - udělat mezi nimi pro člověka nebo počítač čitelný rozdíl je nemožné.

V určitých projektech to může znamenat změnu workflow, přizpůsobení nástrojů verzovatelnosti. Je snazší verzovat několik textových souborů (ve formátu LaTeX) než wordovské dokumenty (na to se musí použít interní nástroj Wordu ... se vším dobrým i zlým).

Odkazy

KAPITOLA 2

GIT

GIT napsal původně Linus Torvalds pro správu zdrojových kódů linuxového jádra. Z licenčních důvodů museli opustit projekt BitKeeper a ruční řešení revizí už nepřicházelo vzhledem k množství změn v úvahu.

GIT je *decentralizovaný* což znamená, že není jeden hlavní repozitář, ke kterému se ostatní "kopie" pouze odvolávají, ale každý *klon* je považován za plnohodnotný repozitář. Všichni mají vždy k dispozici plnou kopii projektu včetně historie - ale mohou mít i několik různých "plných kopií". Správce projektu obvykle sehrává z různých repozitářů změny do hlavního repozitáře a rozhoduje o jejich pořadí a prioritách.

Poznámka: Jak to je v Cleerio: Přes uvedené výše, u většiny projektů používáme *jeden centrální repozitář*, do kterého nahráváme postupně naše změny. Nepoužíváme vlastní kopie celého projektu a navzájem si neklonujeme různé veřejné klony projektu. Příklad: *jachym* vždycky bude stahovat změny z a posílat změny do repozitáře označeného jako *origin* na serveru *gitlab.geo/gp2-lib/*. Nikdy si neudělá kopii od *pavel* nebo *chrudos*. Ale: jachym bude vždycky pracovat minimálně se dvěma klony projektu: s *origin* (tedy tím, co je na serveru) a s *lokální kopií*.

Intenzivně používáme možnost větvení a zpětné sehrávání - každý vývojář je odpovědný za kód, který posílá do hlavní větve. Kód je ale vyvíjen separátně ve větvích a do hlavní větve *masteru* jde až po odladění a když je jisté, že nevzniknou konflikty.

2.1 Vytvoření prázdného lokálního repozitáře

Poznámka: Předpokládáme, že máte nainstalován systém GIT do vašeho počítače.

Pro takové to domácí verzování stačí, když si v adresáři s vaším projektem založíte repozitář.

```
$ mkdir muj_projekt
$ cd muj_projekt
$ git init
```

Příkaz *git init* budete potřebovat po celou dobu životnosti projektu jenom jednou - na jeho začátku. A pak ho zapomenete.

Když si nevíte rady, zkuste nápovědu

```
$ git --help
```

A dostane se vám vyčerpávající odpovědi:

```
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]
           <command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)
   clone      Clone a repository into a new directory
   init       Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)
   add        Add file contents to the index
   mv         Move or rename a file, a directory, or a symlink
   reset      Reset current HEAD to the specified state
   rm         Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
   bisect     Use binary search to find the commit that introduced a bug
   grep       Print lines matching a pattern
   log        Show commit logs
   show       Show various types of objects
   status     Show the working tree status

grow, mark and tweak your common history
   branch     List, create, or delete branches
   checkout   Switch branches or restore working tree files
   commit     Record changes to the repository
   diff       Show changes between commits, commit and working tree, etc
   merge      Join two or more development histories together
   rebase     Forward-port local commits to the updated upstream head
   tag        Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
   fetch      Download objects and refs from another repository
   pull       Fetch from and integrate with another repository or a local branch
   push       Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <command>' or 'git help <concept>'
to read about a specific subcommand or concept.
```

Co vlastně udělal příkaz *git init* ? Vytvořil lokální adresář *.git* s kompletní historií projektu a nějakou tou konfigurací.

Stav vašeho aktuálního lokálního repozitáře získáte příkazem *git status* status může vypadat různě, např.

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

nebo (tyto soubory nejsou v repozitáři a můžu je přidat):

```
$ git status
    On branch prep
    Your branch is up-to-date with 'origin/prep'.
    Untracked files:
```

```
(use "git add <file>..." to include in what will be committed)

    node_modules_ol3/
    package.json.orig
    src/gs/app/gp2.js.orig
    src/gs/layout/layout.js.orig
    src/gs/module/datagrid.js.orig
    src/gs/module/datagrid.js.rej
    ...
```

Status je dobré číst. Pokud změníme nějaký soubor, ukáže nám to status:

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   git.rst

no changes added to commit (use "git add" and/or "git commit -a")
```

Změny mohu uložit do revize pomocí *git commit*. Mohu buď vyjmenovat soubor který chci uložit nebo použít přepínač *-a*, který uloží změny ve všech registrovaných souborech. Další užitečná volba je *-m*, která přidá komentář ke commitu. Pokud nepoužiji volbu *-m*, spustí se textový editor podle nastavení systému, do kterého je potřeba komentář zadat:

```
$ git commit -a -m"Doplnění sekce práce s gitem"
[master 1a99084] Doplnění sekce práce s gitem
1 file changed, 48 insertions(+), 2 deletions(-)
```

git status prozradí, že změny byly uloženy, ale že lokální kopie mého repozitáře je od 2 commity napřed před verzí na serveru *origin*:

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
nothing to commit, working directory clean
```

Příkaz *git show* může ukázat, co bylo konkrétní změnou v poslední (nebo libovolné) revizi (commitu):

```
$ git show
commit 1a9908487f5eebbeaad253e835cabb33ca18a8cd
Author: Jachym Cepicky <jachym.cepicky@gmail.com>
Date:   Mon Sep 19 16:37:50 2016 +0200

    Doplnění sekce práce s gitem

diff --git a/git.rst b/git.rst
index 93cab2e..557bd32 100644
--- a/git.rst
+++ b/git.rst
@@ -97,7 +97,53 @@ A dostane se vám vyčerpávající odpovědi::
     concept guides. See 'git help <command>' or 'git help <concept>'
     to read about a specific subcommand or concept.

-Vytvoření prázdného lokálního repozitáře
```

```
-=======================================
 Co vlastně udělal příkaz `git init` ? Vytvořil lokální adresář `.git` s
 kompletní historií projektu a nějakou tou konfigurací.
+
+Stav vašeho aktuálního lokálního repozitáře získáte příkazem `git status` status
+může vypadat různě, např. ::
+
```

Každá revize má unikátní identifikátor, který mohu kdykoliv v budoucnu použít a vrátit se k němu, často stačí pouze pár unikátních prvních znaků:

```
$ git show 1a9908487f
commit 1a9908487f5eebbeaad253e835cabb33ca18a8cd
Author: Jachym Cepicky <jachym.cepicky@gmail.com>
Date:   Mon Sep 19 16:37:50 2016 +0200

    Doplnění sekce práce s gitem

diff --git a/git.rst b/git.rst
index 93cab2e..557bd32 100644
```

Nový soubor a adresář můžeme přidat (registrovat) v repozitáři příkazem *git add*:

```
$ git add cheatsheet.rst
$ git status

On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   cheatsheet.rst
```

A to je vlastně celé workflow: Pracujete, děláte změny v souborech, ucelené bloky změn ukládáte do repozitáře do samostatných revizí (*git commit*), nové soubory registrujete v repozitáři jak přicházejí (pomocí *git add*).

KAPITOLA 3

Logování

Práce se pracovním záznamem (logem) z gitu je trochu magie. Na štěstí existuje několik základních parametrů a pomůcek pro jednodušší orientaci, na zbytek je Google.

Záznam práce ukáže *git log*:

```
$ git log
commit 1a9908487f5eebbeaad253e835cabb33ca18a8cd
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Mon Sep 19 16:37:50 2016 +0200

    Doplnění sekce práce s gitem

commit bc12c5de2f90ae8832743f5ee8584640b3a59b65
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Thu Sep 15 14:17:38 2016 +0200

    last note

commit 6574bcfd6f1e5b12c57760ed78b545154d666c54
Merge: 7c75607 83221c8
Author: Jachym Cepicky <jachym@users.noreply.github.com>
Date:   Fri Sep 16 14:35:05 2016 +0200

    Merge pull request #1 from madlenkk/master

    Opravy překlepů

commit 83221c8504b9c406057bffcd793d36c7eed5ad93
Author: Magdalena Kabatova <magdalena.kabatova@gmail.com>
Date:   Fri Sep 16 14:29:54 2016 +0200

    Opravy překlepů

commit 7c75607f8faef75ceef6e8349359d66c425e0f02
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Wed Sep 7 16:25:24 2016 +0200

    neco o gitu

commit c9aa4cf1ca1f626ae0566e7eaaa59c714a5e17e6
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Tue Sep 6 09:56:01 2016 +0200
```

```
    pridavam README

commit a7440f4c16ce5473adcecf3c3595b876d997a039
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Tue Sep 6 09:51:22 2016 +0200

    initial commit
```

Ukáže všechny commity v dané větvi, jak šel čas "nekonečně" hluboko do historie. Tento výpis není nejpřehlednější a lze ho různě podifikovat. Lze mimo jiné speficikovat určité časové obdoví - to může být definováno časem nebo rozsahem identifikátorů, např. poslední 2 revize:

```
$ git log -2
commit 1a9908487f5eebbeaad253e835cabb33ca18a8cd
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Mon Sep 19 16:37:50 2016 +0200

    Doplnění sekce práce s gitem

commit bc12c5de2f90ae8832743f5ee8584640b3a59b65
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Thu Sep 15 14:17:38 2016 +0200

    last note
```

ohraničení 2mi revizemi:

```
$ git log c9aa4cf1ca1f626ae05..83221c8504b9c40
commit 83221c8504b9c406057bffcd793d36c7eed5ad93
Author: Magdalena Kabatova <magdalena.kabatova@gmail.com>
Date:   Fri Sep 16 14:29:54 2016 +0200

    Opravy překlepů

commit 7c75607f8faef75ceef6e8349359d66c425e0f02
Author: Jachym Cepicky <jachym.......@mail.com>
Date:   Wed Sep 7 16:25:24 2016 +0200

    neco o gitu
```

Copak zatím dělala *Magdalena*:

```
$ git log --author Magdalena
commit 83221c8504b9c406057bffcd793d36c7eed5ad93
Author: Magdalena Kabatova <magdalena.......@mail.com>
Date:   Fri Sep 16 14:29:54 2016 +0200

    Opravy překlepů
```

3.1 git tree

Lze trochu zpřehlednit výstup:

```
$ git log --oneline --decorate --all --graph

* 1a99084 (HEAD -> master) Doplnění sekce práce s gitem
* bc12c5d last note
```

```
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
|\
| * 83221c8 Opravy překlepů
|/
* 7c75607 neco o gitu
* c9aa4cf pridavam README
* a7440f4 initial commit
```

Poslední příkaz používám jako tzv. *alias* - zkratku - kterou spouštím pomocí *git tree*. Ukazuje mi postup prací, kde jsem já (*HEAD*), kde je server (většinou *origin*), jak spolu souvisí různé větve a podobně. Alias můžete přidat do konfiguračního souboru gitu uloženého někde jako *$HOME/.gitconfig* nebo prostě příkazem *git config*:

```
$ git config --global alias.tree 'log --oneline --decorate --all --graph'

$ git tree
```

3.2 TIG

Pro příkazovou řádku existuje i příkaz *tig*, který dává také celekem přehledný výstup:

```
2016-09-19 20:19 Jachym Cepicky     o [master] {origin/master} dalsi tipy
2016-09-19 20:16 Jachym Cepicky     o doplneni cheatsheetu
2016-09-19 20:00 Jachym Cepicky     o remote
2016-09-19 18:28 Jachym Cepicky     M- Merge branch 'pokusna_vetev'
2016-09-19 17:51 Jachym Cepicky     | o [pokusna_vetev] commit do jine vetve
2016-09-19 17:54 Jachym Cepicky     o | vyroba konfliktniho řádečku
2016-09-19 17:43 Jachym Cepicky     o- pokracovani dokumentace
2016-09-19 16:37 Jachym Cepicky     o Doplnění sekce práce s gitem
2016-09-15 14:17 Jachym Cepicky     o last note
2016-09-16 14:35 Jachym Cepicky     M- Merge pull request #1 from madlenkk/master
2016-09-16 14:29 Magdalena Kabatova | o {madlenkk/master} Opravy překlepů
2016-09-07 16:25 Jachym Cepicky     o- neco o gitu
2016-09-06 09:56 Jachym Cepicky     o pridavam README
2016-09-06 09:51 Jachym Cepicky     I initial commit
```

KAPITOLA 4

Grafické uživatelské rozhranní

Není moc lidí, co si vystačí se základní sadou příkazů v konzoli a k tomu několika příhodnými aliasy. Lidé chtějí změny vidět. Různá vývojová prostředí a textové editory obsahují sadu nástrojů pro práci s GIT a pro vizualizaci změn.

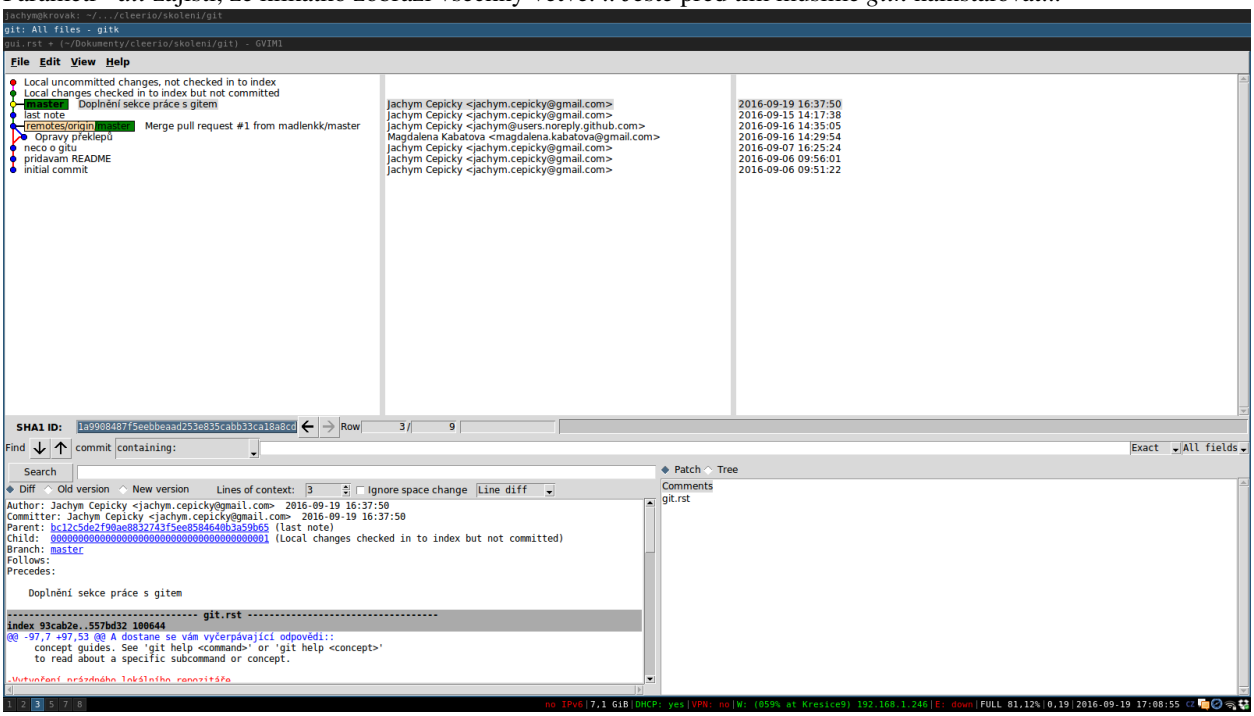
Samozřejmě lze doporučit zejména textový editor VIM a rozšíření Fugitive. VIM si určitě nastavte jako editor pro řešení konfliktů

```
$ git config --global diff.tool 'gvimdiff'
$ git config --global merge.tool 'gvimdiff'
```

Pro ty z vás, kteří se s VIMem moc nekamarádí, lze doporučit program *gitk*, který používá dokonce i Pavel. Spouští s příkazem

```
$ gitk --all
Program 'gitk' v současnosti není nainstalován. Můžete jej nainstalovat napsáním:
sudo apt install gitk
```

Parametr *–all* zajistí, že klikátko zobrazí všechny větve. .. Ještě před tím musíme *gitk* nainstalovat...



Jak vidíte, grafické rozhraní nečiní práci s gitem nijak předhlednější ani jednodušší natož příjemnější.

KAPITOLA 5

Větvení

Tohle si necháme na praktickou ukázku, protože se to s ní pochopí líp. Větvení vám umožní "jakoby" vytvářet kopie adresáře současné práce do zálohy, něco vyzkoušet, pak se vrátit k předchozí verzi, začít jinak, zkusit to na jiné větvy zas jinak atp.

Další možný scénář je, že dva lidé pracují na různých věcech, každý ve své větvi, navzájem si "nelezou do souborů", a nakonci se jejich práce sehraje dohromady.

GIT by vás měl povzbuzovat v tom, abyste vytvářeli *co nejvíce větví*. V rámci Cleerio vývojáři vytváří samostatnou větev na každý "ticket" v Redmine. Teprve když je větev schválena, dává se do hlavní větve, která se jmenuje *master*.

Pro práci s větvemi slouží *git branch* a umí toho hodně.

Seznam dostupných větví - na serverech i lokálních:

```
$ git branch -la

* master
remotes/origin/master
```

Tento výpis znamená:

- na serverech mohou být jiné větve, než u vás na lokále.
- můžete mít různě pojmenované větve u sebe a na serveru - a přesto budou korespondovat

Jak to vypadá na frontendu naší aplikace:

```
$ git branch -la
...
remove
setOptions_refactoring
sprint-03
templates
tmp
tmp2
remotes/origin/1397_maxExtent
remotes/origin/1612_neighbours
remotes/origin/2422_faster_vectors
remotes/origin/2478_rotateButton
remotes/origin/3476_renameCallbackListen
remotes/origin/3971_jsdoc
remotes/origin/4433_iefixes
remotes/origin/4537_partialLV
...
```

Jak to vypadá v PyWPS:

```
...
  help
* master
  pygrass
  pywps-3.2
  pywps-3.2.3
  pywps-4
  web
  ...
  remotes/jachym/1.0
  remotes/jachym/1.0@1
  remotes/jachym/109_close_stream
  remotes/jachym/122_documentation
  ...
  remotes/jan-rudolf/1.0
  remotes/jan-rudolf/1.0@1
  remotes/jan-rudolf/flask
  remotes/jan-rudolf/gh-pages
  remotes/jan-rudolf/master
  ...
  remotes/origin/1.0
  remotes/origin/1.0@1
  remotes/origin/151_dblog2
  remotes/origin/HEAD -> origin/master
  remotes/origin/flask
  remotes/origin/master
```

Jsou vidět tři registrované vzdálené servery: *origin*, *jachym* a *jan-rudolf*, každý s hromadou větví.

GIT umí udržet pořádek v tom, jaká větev na lokále odpovídá jaké větvi na serveru. Větev můžete smazat z lokálu - ale na serveru zůstane. Můžete ji smazat i ze serveru. Můžete ji na lokále přejmenovat - ale na serveru zůstane stejná.

Založení nové větve je celkem primocare:

```
$ git branch pokusna_vetev
```

Ověříme jaké máme větve:

```
$ git branch -la

* master
pokusna_vetev
remotes/origin/master
```

Ověříme, *kde* v historii větev vlastně vznikla (buď v *gitk* nebo pomocí logu):

```
$ git tree

* 1a99084 (HEAD -> master, pokusna_vetev) Doplnění sekce práce s gitem
* bc12c5d last note
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
|\
| * 83221c8 Opravy překlepů
|/
* 7c75607 neco o gitu
* c9aa4cf pridavam README
* a7440f4 initial commit
```

Vidíte, že *pokusna_vetev* vznikla na místě, kd se aktuálně nachází moje rozdělaná práce (*HEAD*), což shodou okolností

odpovídá stavu větve *master*.

Přepnu se do větve *pokusna_vetev* a zapíšu všechny změny:

```
$ git checkout pokusna_vetev

$ git commit -a -m"Commit do jine vetve"

$ git tree

* 2e03719 (HEAD, pokusna_vetev) commit do jine vetve
* b89a5c0 (master) pokracovani dokumentace
* 1a99084 Doplnění sekce práce s gitem
* bc12c5d last note
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
|\
| * 83221c8 Opravy překlepů
|/
* 7c75607 neco o gitu
* c9aa4cf pridavam README
* a7440f4 initial commit
```

Přepnu se zpátky do větve *master* a provedu záměrně nějakou konfliktní změnu v existujícím souboru:

```
$ git checkout master

# editace existujícího souboru

$ git commit -m"vyroba konfliktniho řádečku" -a
$ git show

commit 096304c55364e3e4b849fe567402b3444258a49e
Author: Jachym Cepicky <jachym.cepicky@gmail.com>
Date:   Mon Sep 19 17:54:14 2016 +0200

    vyroba konfliktniho řádečku

diff --git a/vetveni.rst b/vetveni.rst
index b754523..29faf4d 100644
--- a/vetveni.rst
+++ b/vetveni.rst
@@ -72,4 +72,4 @@ Jak to vypadá v PyWPS::
       remotes/jan-rudolf/master


-
+Tady vyrobíme nějaký ten konfliktní řádeček
```

Jak to vypadá s historií revizí:

```
$ git tree

* 096304c (HEAD -> master) vyroba konfliktniho řádečku
| * 2e03719 (pokusna_vetev) commit do jine vetve
|/
* b89a5c0 pokracovani dokumentace
* 1a99084 Doplnění sekce práce s gitem
* bc12c5d last note
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
|\
```

```
| * 83221c8 Opravy překlepů
|/
* 7c75607 neco o gitu
* c9aa4cf pridavam README
* a7440f4 initial commit
```

A nyní můžeme vyzkoušet spojení větví. Chci změny **z větve** *pokusna_vetev* spojit **do větve** *master*:

```
# projistotu

$ git checkout master
$ git status
On branch master
Your branch is ahead of 'origin/master' by 3 commits.
  (use "git push" to publish your local commits)
 ...

# vlastní merge

$ git merge pokusna_vetev

Auto-merging vetveni.rst
CONFLICT (content): Merge conflict in vetveni.rst
Automatic merge failed; fix conflicts and then commit the result.
```

Hurá, máme konflikt a musíme ho vyřešit. Protože máme perfektní nástroj na řešní konfliktů nastavený, stačí použít *git mergetool*, který spustí textový editor VIM, ve kterém mohu pohldně konflikty vyřešit.:

```
$ git mergetool
```

Merge je v podstatě samostaný commit, po vyřešení konfliktů musím commit vyrobit ručně:

```
$ git commit
```

V textovém editoru se mi nabídne komentář "Merge branch 'pokusna_vetev"'. Uložte soubor a mergování bude samo automaticky pokračovat.

Výsledek:

```
$ git tree
*   6ebd5ea (HEAD -> master) Merge branch 'pokusna_vetev'
|\
| * 2e03719 (pokusna_vetev) commit do jine vetve
* | 096304c vyroba konfliktniho řádečku
|/
* b89a5c0 pokracovani dokumentace
* 1a99084 Doplnění sekce práce s gitem
* bc12c5d last note
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
...
```

Mám novou revizi 6ebd5ea, vzniklou sloučením větví *pokusna_branch* a *master*, vyřešili jsme ručně konflikt. Vše se děje na lokálním repozitáři - serverová verze pořád zůstala *zamrzlá* na revizi 6574bcf. Je čas poslat změny na server.

KAPITOLA 6

Vzdálené repozitáře

Vzdálený repozitář si opatříme příkazem *clone*:

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

Nebo v existujícím repozitáři si ho můžeme přidat příkazem *remote*:

```
$ git remote add madlenkk https://github.com/madlenkk/git.git
```

Protože kolegové nikdy nespí, je potřeba často a pravidelně stahovat změny z repozitáře. Jako nejlepší praktika se nám osvědčil systém *git fetch - git rebase*.

Poznámka: Často se v návodech objevuje postup *git pull –rebase*, ten má ale tu nevýhodu, že v jednom kroku musíte řešit případné konflikty mezi verzí na server a verzí ve vašem lokálním repozitáři. *git fetch* vám pouze stáhne změny ze serveru, ale nechá vas pracovat tam "kde zrovna jste". *git rebase* vám pak pomůže při "posunu nad" aktuální konec větve ze serveru.

Stánutí všech revizí ze všech serverů:

```
$ git fetch --all
Fetching origin
Fetching madlenkk
From https://github.com/madlenkk/git
 * [new branch]      master     -> madlenkk/master
```

git tree ukáže, kde jsem:

```
$ git tree

*   6ebd5ea (HEAD -> master) Merge branch 'pokusna_vetev'
|\
| * 2e03719 (pokusna_vetev) commit do jine vetve
* | 096304c vyroba konfliktniho řádečku
|/
* b89a5c0 pokracovani dokumentace
* 1a99084 Doplnění sekce práce s gitem
* bc12c5d last note
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
|\
| * 83221c8 (madlenkk/master) Opravy překlepů
|/
* 7c75607 neco o gitu
```

```
* c9aa4cf pridavam README
* a7440f4 initial commit
```

Vidíte, že magdalenina větev *master* je o revizi za serverem *origin* a ten je zase o nutný kus historie za mou lokální větví.

6.1 push a pull (vlastně fetch)

Pro práci se vzdálenými serveri slouží dva základní příkazy *git push* a *git pull*. Nám se ale osvědčilo používat místo *pull* kombinaci *fetch* a *rebase*.

Rozdíl mezi *fetch* a *pull* je, že *'git pull' udělá v jednom kroce 'git fetch' následovaný 'git merge'. pull* vás dounutí okamžitě řešit konflikty mezi lokální a vzdálenou větví.

git fetch vám dovolí pouze stáhnout nové revize ze vzdáleného serveru. *git rebase* vám umožní se se svými lokálními commity *posunout nad* revize stažené ze serveru.

Pokud chci poslat změny na vzdálený server, je to už jednoduché, použiji *git push*.

Takže nejpre stáhneme změny ze vech vzdálených repozitářů:

```
$ git fetch --all
```

Podíváme se, kde je kdo, kde je jaká větev:

```
$ git tree

*   6ebd5ea (HEAD -> master) Merge branch 'pokusna_vetev'
|\
| * 2e03719 (pokusna_vetev) commit do jine vetve
* | 096304c vyroba konfliktniho řádečku
|/
* b89a5c0 pokracovani dokumentace
* 1a99084 Doplnění sekce práce s gitem
* bc12c5d last note
*   6574bcf (origin/master) Merge pull request #1 from madlenkk/master
|\
| * 83221c8 (madlenkk/master) Opravy překlepů
|/
* 7c75607 neco o gitu
```

Vidíme, že jsme daleko před verzí v repozitáři *origin* in *madlankk*. Konflikty nevidět, mohu vesele poslat svoje lokální revize na server *origin*:

```
$ git push origin
Username for 'https://github.com': jachym
Password for 'https://jachym@github.com':
Counting objects: 25, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (25/25), done.
Writing objects: 100% (25/25), 120.36 KiB | 0 bytes/s, done.
Total 25 (delta 14), reused 0 (delta 0)
remote: Resolving deltas: 100% (14/14), completed with 3 local objects.
To https://github.com/geosense/git.git
   6574bcf..6ebd5ea  master -> master
jachym@krovak:~/.../cleerio/skoleni/git (master)$
```

A zkontrolovat, jak se věci mají pomocí *git tree*:

```
$ git tree
*   6ebd5ea (HEAD -> master, origin/master) Merge branch 'pokusna_vetev'
...
```

Vidíte, že větve *master* a *origin/master* jsou již v souladu.

Nyní přijde ale do práce *madlenkka* a co nevidí - origin jí "ujel" o pěkných pár commitů. Co jí zbývá, než stáhnout změny ze serveru:

```
$ git fetch --all
```

Posunout svoji aktuální práci *nad* verzi na serveru:

```
$ git rebase origin/master
```

KAPITOLA 7

Cheatsheet

git init inicializace repozitáře

git clone vytvoření lokální kopie ze vzdáleného repozitáře

git commit Uložení lokálních změn do lokálního repozitáře

Asi nejdůležitější a nejčastěji používaný příkaz

git branch Vytváření nové větve (správa větví)

git merge sloučení změn *z druhé větve do aktuální*

git checkout Přepnutí se do jiné branche, na jiný commit, na jiný tag

git fetch Stažení změn ze serveru

git rebase Přeskládání aktuální práce nad nový základ (např. nad změny stažené před tím ze serveru pomocí *git fetch*)

git push poslání lokálních změn na vzdálený server

git pull alternativní stažení změn ze serveru spolu s *mergováním* konfliktů mezi lokální a vzdálenou větví

git remote správa vzdálených repozitářů

KAPITOLA 8

Typické workflow

8.1 Bez větví

Klonování existujícího repozitáře:

```
git clone https://github.com/geosense/git
```

Na začátku práce stáhnout aktuální verze ze vzdálených repozitářů a přeskládat svoji případnou práci nad ně:

```
git fetch --all
git rebase
```

Následuje práce v textovém editoru, a mnoho a mnoho commitů:

```
git commit -a -m"důležitá změna"
git commit -m"jiná změna - ale jenom v jednom souboru" gui.rst
```

Po skončení směny - nebo v případě požáru:

```
git push
```

A opusťte budovu

Když se chci podívat, co se změnilo, kde jsem:

```
git status

git tree
```

8.2 když pracuji s větvemi

vyrobím a přepnu se do větve:

```
git branch jmeno_vetve
git checkout jmeno_vetve
```

Dál pracuji normálně:

```
git commit
git commit
...
```

```
git push
git fetch --all
git rebase

git commit
git commit
git commit
...
```

Sem tam použiji *git tree* a *git status* (nebo holt *gitk*, no...)

Když chci sloučit změny *do cílové větve 'master'*:

```
git checkout master
git merge jmeno_vetve
git push
```

KAPITOLA 9

Dalších pár tipů

9.1 Tagy

tag je záložka posazená na určitou revizi. Trochu se tváří jako větev. Je to místo, kam se můžete chtít v budoucnu vrátit (třeba verze):

```
git tag verze-1.2.3
```

9.2 Hooky

Hook (háček) je, co se má stát, když se v repozitáři udělá nějaká změna. Nejčastěji na *push* do repozitáře, ale může to být i nová větev, tag, smazání větve, atd atd.. My na každý *push* pouštíme sadu testů a build aplikace

9.3 Rozdíl mezi GIT, GITHub, GitLab

GIT je systém pro správu verzí, umožňující vedení několika paralelních větví projektu, umožňuje jejich slučování, vede historii projektu. GIT je open source.

GITHub je webová stránka/služba, která dává k dispozici zdarma git repozitáře na serveru (pro open source projekty), přidává sociální aspekt k vývoji, bug tracker, wiki, spoustu webových nastavovátek. GITHub sice hostuje hromadu open source projektů, ale sama o sobě je proprietární

GITLab Je trochu jako GITHub, ale open source, můžete si to nainstalovat k sobě na server a provozovat. U nás žije na http://gitlab.geo doméně

Odkazy na zdroje

- Distribuovaná správa revizí s GIT.

License

_static/cc-by-sa.png	Text je licencován pod Creative Commons Attribution-ShareAlike 4.0 International License.

Indices and tables

- genindex
- modindex
- search