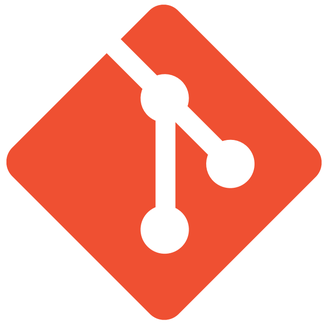
Lynt SERVICES



git

Základní školení

infrastruktura | webová řešení | marketing

Cíl

Seznámit se základními principy verzovacích systémů, s verzovacím systémem Git a jeho základními vlastnostmi. Osvojit si základní příkazy a pracovní postupy na praktických příkladech.

Osnova školení

1. **Úvod do problematiky VCS a Git.**

   Základní principy a terminologie, specifika Gitu.

2. **Instalace a počáteční nastavení.**

   Instalační balíček, práce z příkazové řádky, nápověda.

3. **Práce s lokálními repositáři.**

   Vytvoření repositáře, práce se soubory, ignorované soubory, procházení historie.

4. **Vzdálené repositáře.**

   Typy vzdálených repositářů, správa repositářů, stahování a nahrávání změn.

Osnova školení

5. **Správa tagů.**

   Typy tagů, jejich vytváření a nahrávání do vzdálených repositářů.

6. **Využití větví.**

   Základní principy, vytváření větví, přepínání, slučování větví, mazání. Řešení konfliktů.

7. **Práce se stash.**

   Principy, ukládání, obnovování a odstraňování rozpracovaných změn.

8. **Vzdálené větve.**

   Rozdíl mezi lokálními a vzdálenými větvemi, operace nahrávání a stahování změn.

Osnova školení

9. **Workflow.**

   Větve pro orgranizaci práce: feature branches, git flow a github flow.

10. **Patche.**

    Vytváření a aplikování patchů.

11. **Referencování commitů.**

    Odkazování na commity a skupiny commitů.

12. Závěr

1.

Úvod do problematiky VCS a Git.

Co je VCS?

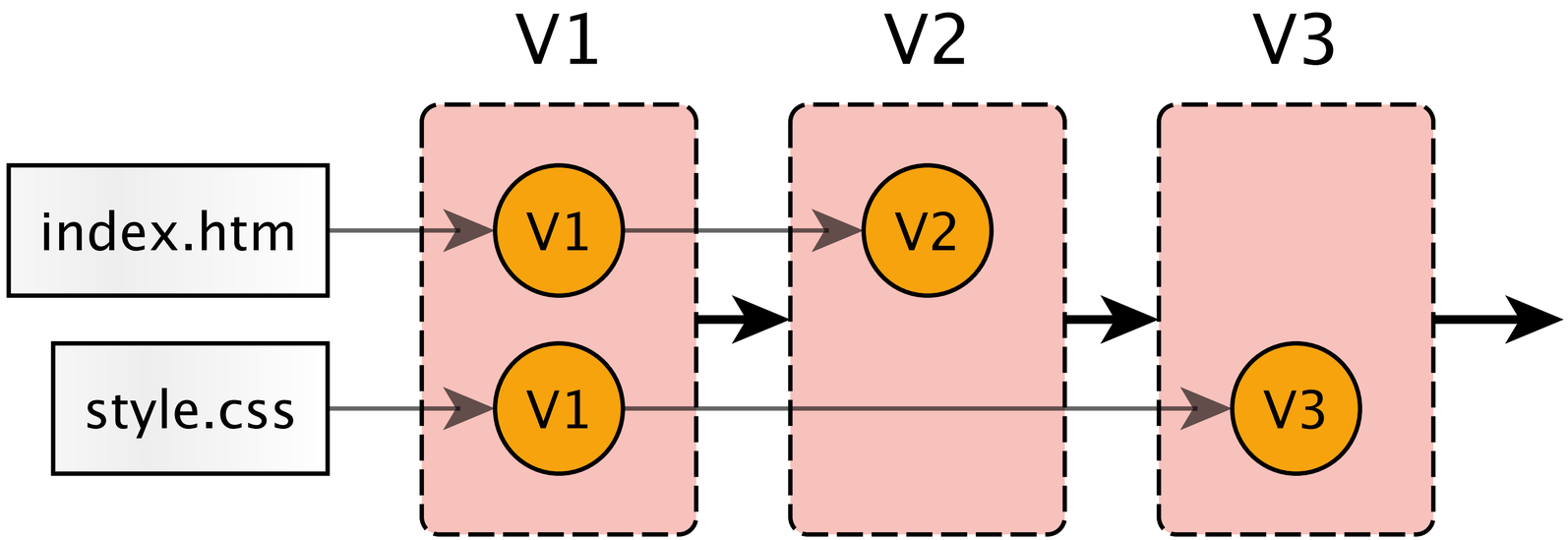
- **Version Control System** (systém pro správu verzí).
- Software, který umožní zaznamenávat změny v určité sadě souborů tak, jak probíhaly v čase.
- Umožňuje typicky obnovu starších verzí, procházení a kontrolu provedených změn, zjištění, kdo změnu provedl a další.
- Usnadňuje spolupráci více lidí na jednom projektu.

Základní pojmy

- **Version (Revision)** – uložený stav souborů tak, jak vypadaly v určitém okamžiku. Někdy též nazývaný **Commit** (podst. jméno).
- **Repository** – databáze obsahující historii změn a všech dat s ní souvisejících.
- **Working Copy** – kopie určité verze souborů z repositáře. Typicky uložena na lokálním souborovém systému, aby bylo možné nad ní provádět změny.
- **Check Out** – stažení určité verze z repositáře a vytvoření working copy.
- **Commit (Check In)** – uložení working copy do repositáře jako nové verze.

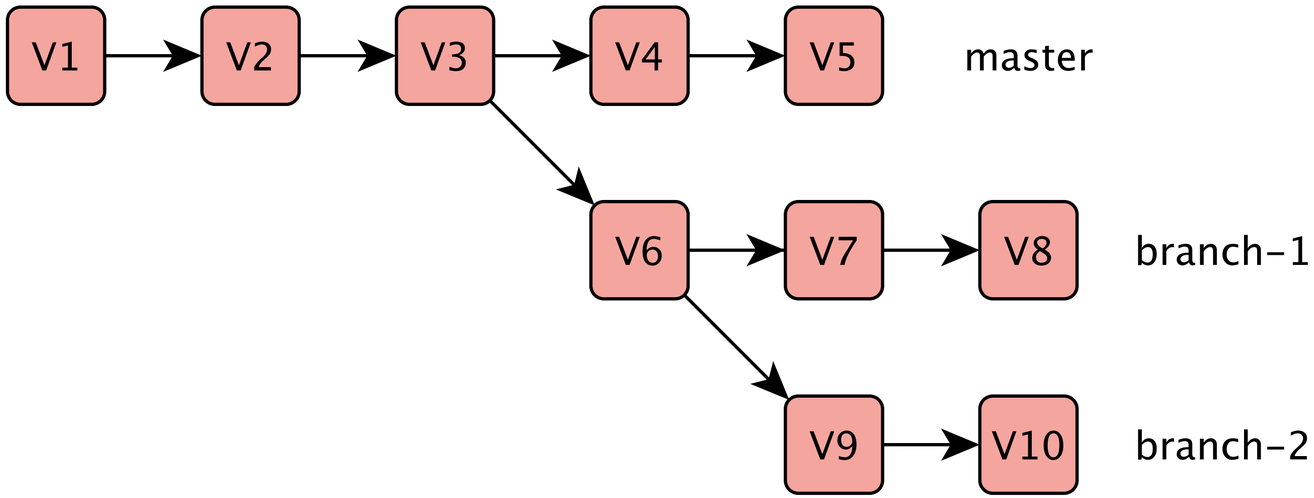
Codeline

- Vývojová linie sady souborů.



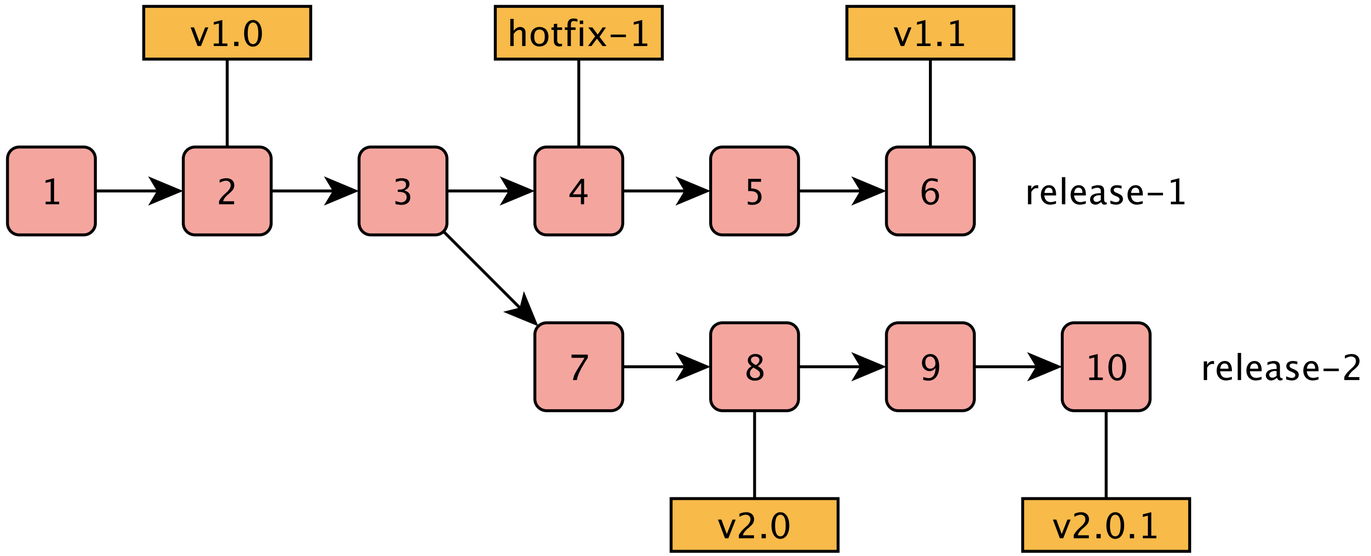
Branch

- Vývojová linie, která je vychází z jiné a je vyvíjena nezávisle na původní.



Pojmy k codeline

- **Tag (Label)** – pojmenovaná reference na určitou verzi.
- **Tip (Head)** – poslední verze v dané codeline.



Typy VCS

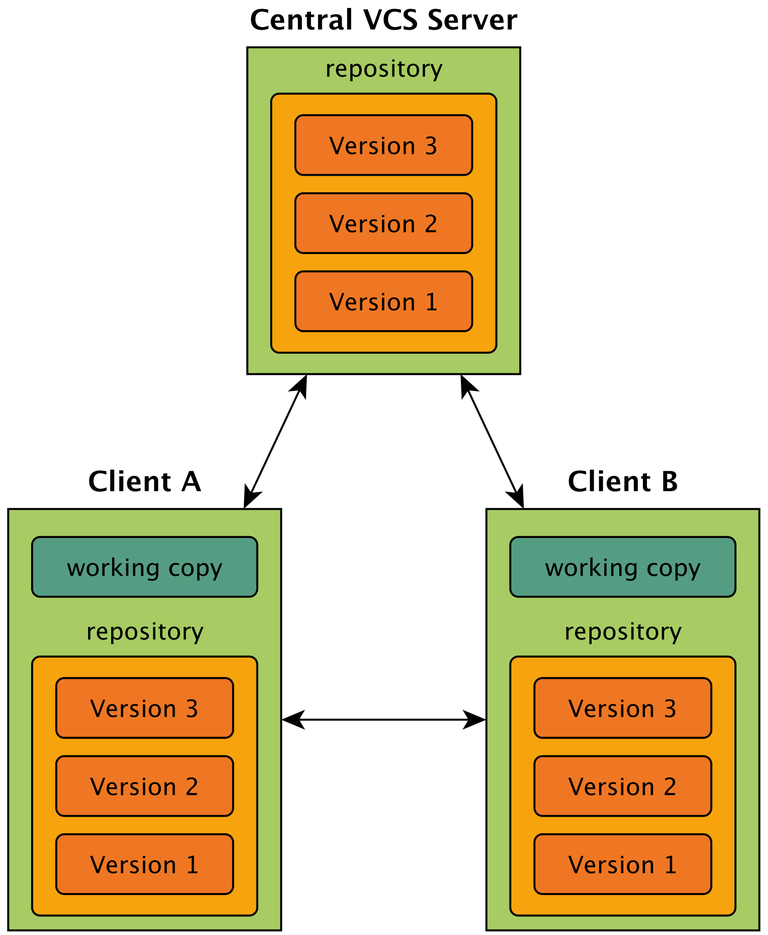
- **Lokální** – repository je uložena na lokálním stroji. Všechny operace probíhají lokálně.
- **Centralizované** – repository je uložena na centrálním serveru, lokálně pak pouze pracovní kopie. K operacím s repository je potřeba centrální server.
- **Distribuované** – repository je uložena v každém uzlu, každý klient má svojí kopii historie. Většina operací probíhá lokálně a je potřeba je synchronizovat mezi klienty.

Git

- http://git-scm.com/
- Distribuovaný VCS
- Důraz na:
  - rychlost,
  - jednoduchost,
  - vysoký počet paralelních větví,
  - efektivitu.
- Vznikl v roce 2005 za účelem vývoje Linuxového kernelu.

Distribuovaný VCS

- Každý uzel má kompletní kopii repositáře.
- Klient pracuje se svojí **Working Copy**, většina operací s repositářem probíhá lokálně, včetně ukládání změn.
- Změny mezi uzly je možné synchronizovat:
  - typicky prostřednictvím centrálního serveru,
  - klienti se mohou synchronizovat mezi sebou.



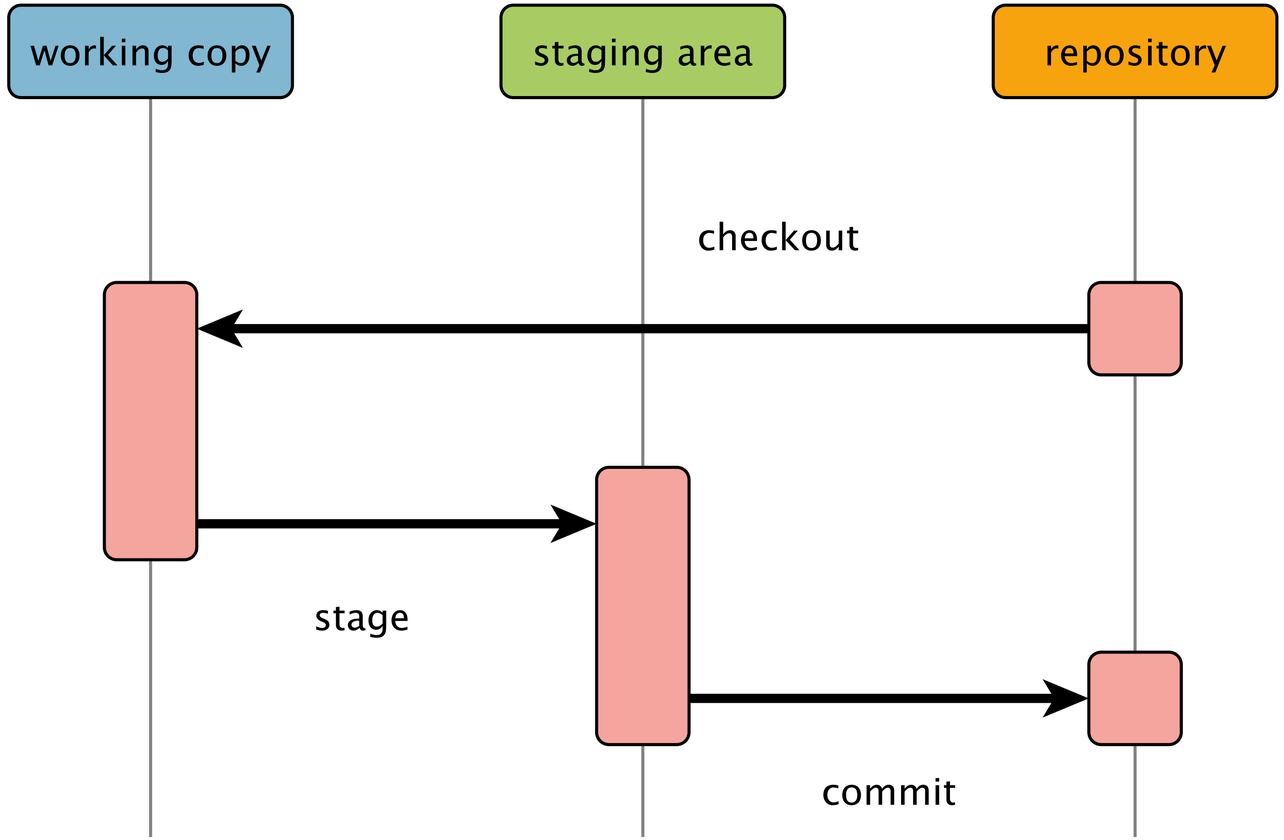
Odlišnosti v pojmech

HEAD – označuje commit, který je aktuálně checked-out ve Working Copy – typicky vrchol aktuální větve.

Rozdělení souborů

- Soubory ve Working Copy jsou rozděleny na:
  - **Commited** – soubor je uložen v repository (lokálním).
  - **Modified** – soubor je modifikován, ale není uložen v repository.
  - **Staged** – modifikovaný soubor, který je označen ke commitu.

Rozdělení lokálního repository



- Do repository se uloží **pouze změny**, které **byly připraveny ve staging area!**
- Změny se nejprve **uloží pouze lokálně**, nikoliv na centrální server. **Nutno synchronizovat ručně!**

Označení objektů

Commity a jiné objekty jsou označeny pomocí **SHA-1 checksums**. Každý objekt je tak jednoznačně identifikován. Ve výpisech jsou zobrazeny v hexadecimálním tvaru:

```
46a5f3c6f680e4feda9b17e9b9439947efc34f5a
```

Při odkazování není nutné psát celý checksum, ale stačí nejmenší unikátní prefix, nejméně však 7 znaků:

```
46a5f3c
```

2.

Instalace a počáteční nastavení.

Instalace

Windows

http://msysgit.github.io/

Linux

z balíčků distribuce

Mac OS X

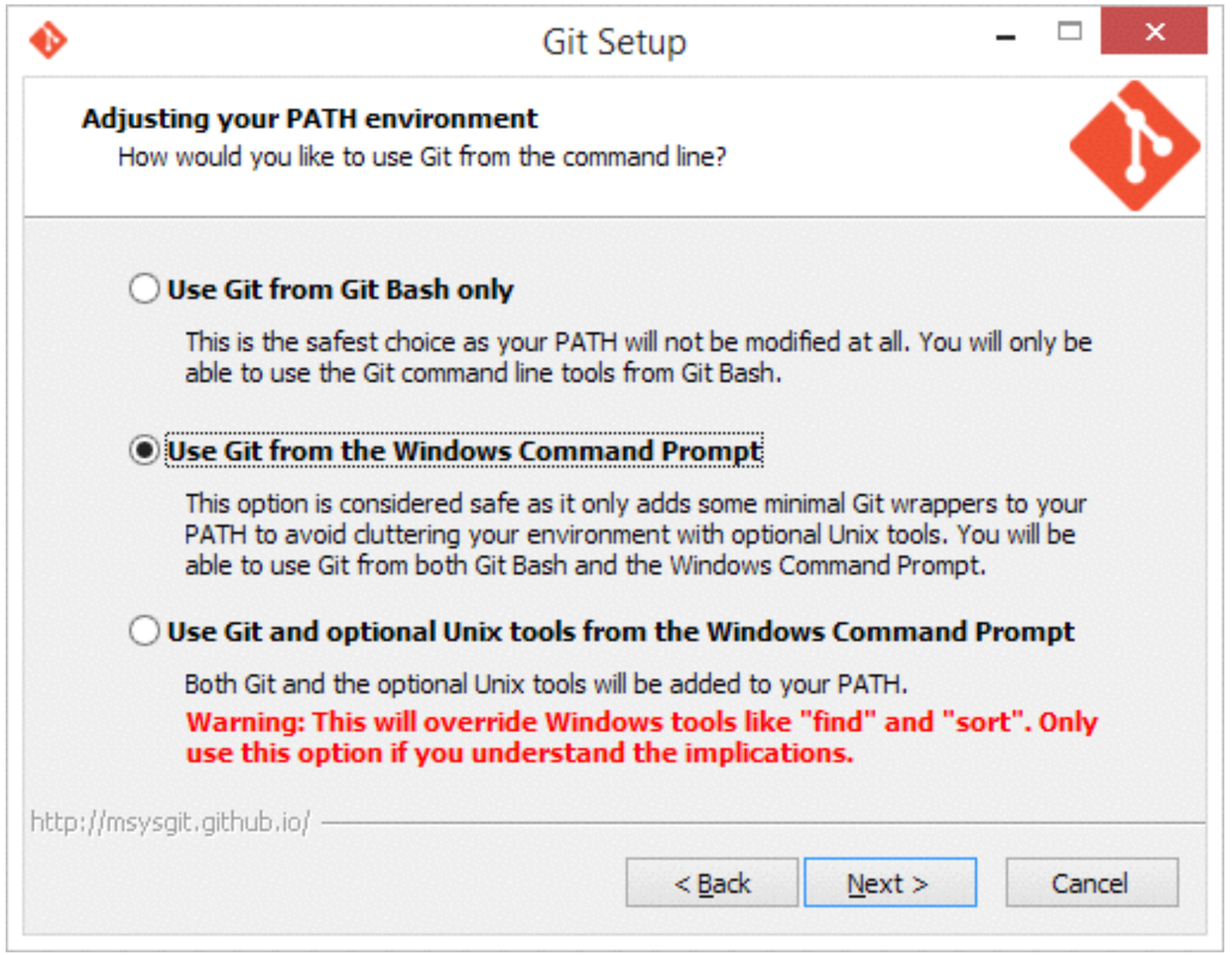
s XCode, nebo Homebrew:

```
brew install git
```

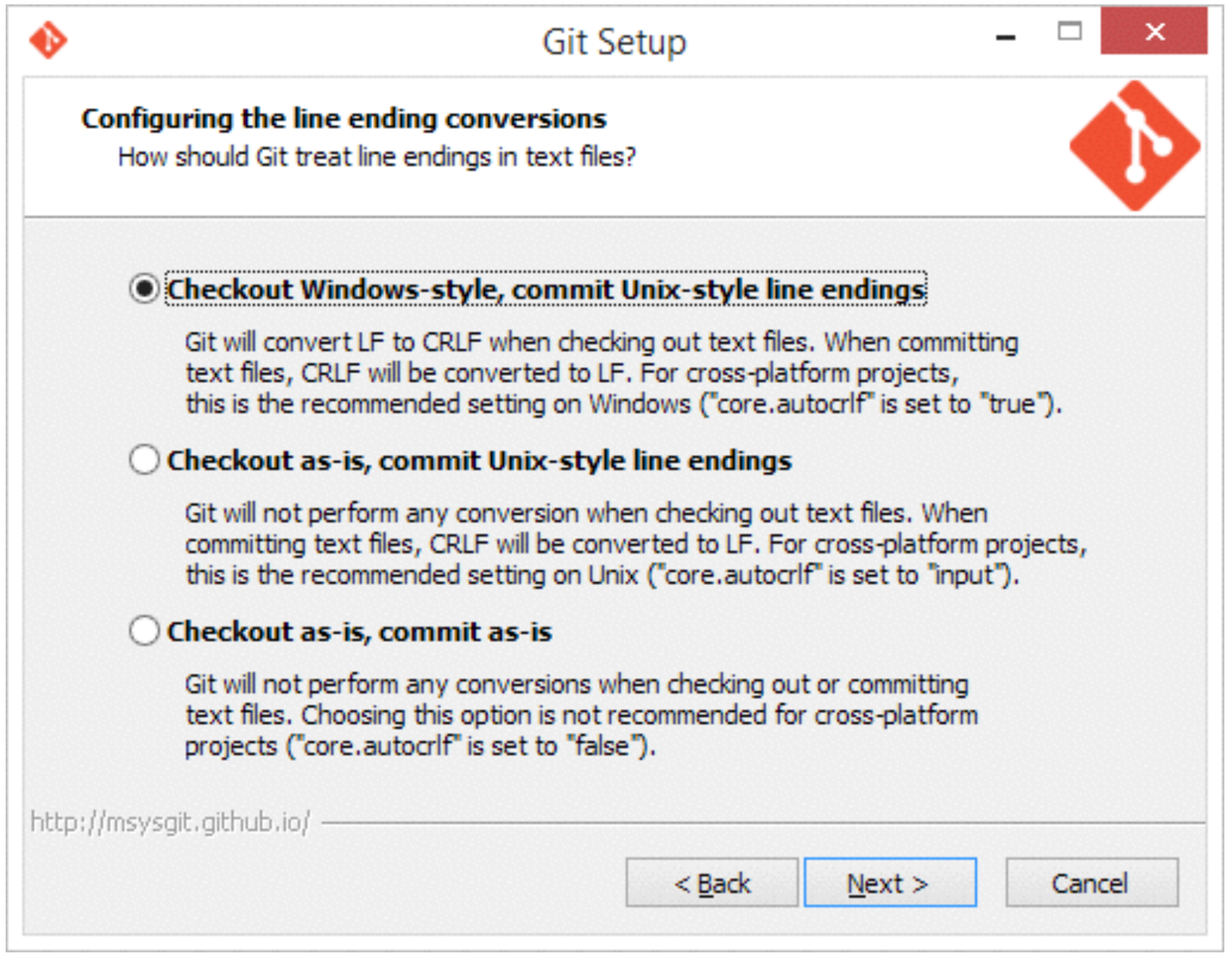
Praktická část

Instalace Git.

Instalace – nastavení PATH



Instalace – nastavení CRLF



Praktická část

Nastavení a nápověda.

Spuštění – git bash

Nastavení uživatele

```
git config --global user.name "Jan Smitka"
git config --global user.email "jan.smitka@lynt.cz"
```

Zobrazení aktuálního nastavení

Globální nastavení

```
git config --global --list
```

Uloženo v `~/.gitconfig`

Nastavení v aktuálním repositáři

```
git config --list
```

Uloženo v `$GIT_DIR/config` (tj. ve složce .git, bude vysvětleno dále).

Získání nápovědy

Index nápovědy

```
git help git
```

Nápověda pro příkaz <command>

```
git help <command>
git <command> --help

git help <command> --web
```

3.

Práce s lokálními repositáři.

Lokální repositář

- Lokální repositář je typicky uložen ve working copy – podsložka **.git**.
- Obsahuje kompletní historii, metadata, konfiguraci, nastavení vzdálených větví a další.
- Možno vytvořit dvěma způsoby:
  - Inicializace prázdného repositáře.
  - Naklonování existujícího repositáře.

Inicializace prázdného repositáře

V libovolné složce:

```
git init
```

Příkaz vytvoří v aktuální složce podsložku .git, ve které je prázdný repositář. Aktuální složka je považována za Working Copy, existující soubory jsou Untracked.

Ve vytvořeném repositáři je vytvořen ukazatel na prázdnou větev s názvem master.

Naklonování existujícího repositáře

V libovolné složce:

```
git clone <remote-url>
```

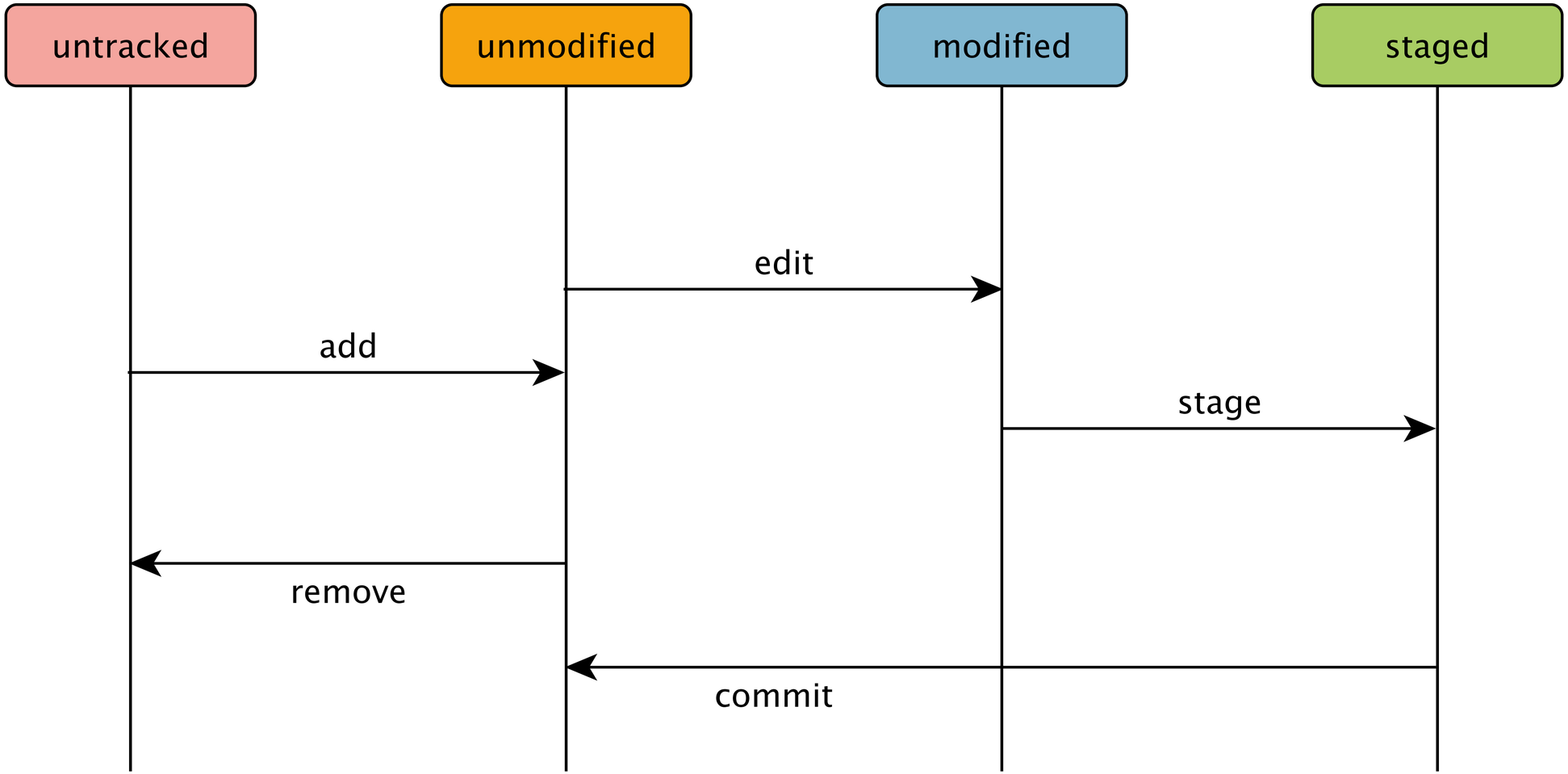
Stáhne repositář z <remote-url>, uloží jej do podsložky s názvem repositáře a vytvoří Working Copy z jeho hlavní větve.

Stavy souborů

Soubory jsou rozděleny do následujících stavů:

- **Untracked** – soubor není gitem spravován.
- **Tracked** – soubor je spravován gitem, tj. byl zaznamenaný v repositáři v posledním snapshotu, nebo připraven ve Staging Area ke commitu. Může dále být:
  - **Unmodified** – nemodifikovaný.
  - **Modified** – modifikovaný.
  - **Staged** – připraven ke commitu.

Životní cyklus souborů



Zjištění stavu souborů

`git status`

```
panda@panda-mac test $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  new file:   CHANGES
  deleted:    OLD_FILE

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:   INSTALL

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

  README
```

Připravení souborů ke commitu

```
git add <file>
```

Přidá soubory do Staging Area (Untracked i Modified).

```
panda@panda-mac test $ git add INSTALL
panda@panda-mac test $ git add README
panda@panda-mac test $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  new file:   CHANGES
  modified:   INSTALL
  deleted:    OLD_FILE
  new file:   README
```

Připravení soubor ke commitu

Ke commitu jsou připraveny soubory v takovém stavu, v jakém byl spuštěn git add.

Pokud se soubor změní, je nutné spustit git add znovu:

```
panda@panda-mac test $ echo "Run rm." >> INSTALL
panda@panda-mac test $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  new file:   CHANGES
  modified:   INSTALL
  deleted:    OLD_FILE
  new file:   README

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:   INSTALL
```

Odstranění souboru ze Staging

```
git reset HEAD <file>
```

```
panda@panda-mac test $ git reset HEAD README INSTALL
Unstaged changes after reset:
M INSTALL
panda@panda-mac test $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  new file:   CHANGES
  deleted:    OLD_FILE

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:   INSTALL

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

  README
```

Odstranění **Tracked** souboru

Po odstranění je též nutné změnu připravit do Staging:

```
panda@panda-mac test $ rm OLD_FILE
panda@panda-mac test $ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  deleted:    OLD_FILE
```

Změnu připravíme také pomocí:

```
git add <file>
```

Odstranění souboru – zkráceně

```
git rm <file>
```

Pokud soubor **existuje**, **smaže jej** a změnu **připraví** do Staging.

Pokud **neexistuje**, ale byl smazán, **připraví** změnu do Staging.

Untracked soubory nelze takto smazat.

Obnova změn

Změny, které dosud nebyly commitnuty, lze zrušit:

```
git checkout -- <file>
```

Soubory obnoví buď do posledního stavu zaznamenaného ve Staging, nebo do stavu v poslední revizi, pokud soubor není ve Staging:

```
panda@panda-mac test $ cat INSTALL
0
panda@panda-mac test $ echo 1 > INSTALL
panda@panda-mac test $ git add INSTALL
panda@panda-mac test $ echo 2 > INSTALL
panda@panda-mac test $ cat INSTALL
2
panda@panda-mac test $ git checkout -- INSTALL
panda@panda-mac test $ cat INSTALL
1
```

Obnova změn

Pokud chceme obnovit změny, které již byly připraveny ve Staging, je potřeba nejprve změnu odebrat ze staging a poté provést checkout:

```
git reset HEAD <file>
git checkout -- <file>
```

Přesun souborů

Git do databáze neukládá přesuny a přejmenování souborů – ukládá nový soubor, smazaný soubor, a přesuny detekuje zpětně.

```
panda@panda-mac test $ mv INSTALL INSTALL.NEW
panda@panda-mac test $ git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

  deleted:    INSTALL

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

  INSTALL.NEW
```

Přesun souborů – příprava změny

Po přesunu nutné přidat obě změny:

```
panda@panda-mac test $ git add INSTALL INSTALL.NEW
panda@panda-mac test $ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  renamed:    INSTALL -> INSTALL.NEW
```

Commit

```
git commit
git commit –m "<message>"
```

Pokud není zadaná **Commit Message** pomocí přepínače -m, spustí se editor (vim).

Změna posledního commitu

Aktuální změny lze sloučit s posledním commitem a ten tak modifikovat:

```
git commit --amend
```

Pokud ve staging nejsou žádné změny, lze použít k přepsání poslední Commit Message.

Pozor! Tímto přepínačem dochází k úpravě historie, což může způsobit problémy, pokud již byla předchozí změna publikována!

Ignorované soubory

Git může některé soubory ignorovat – např. zkompilované soubory, logy, ...

Vytvoření souboru .gitignore s maskami, které mají být ignorovány. Př.:

```
*.log                  # soubory s příponou log
logs/*                 # všechny soubory ve složce logs,
                         podsložky zůstávají
!logs/.htaccess        # .htaccess v logs se neignoruje
temp/                  # temp a všechny soubory
                         v podsložkách
```

Ignorované soubory

Soubor .gitignore platí pro adresář a všechny jeho podsložky – možno mít více .gitignore v jednom repository.

Git ukládá pouze soubory. Složka, která je prázdná, nebo jejíž všechny soubory jsou ignorovány, se do repository neuloží.

Praktická část

Práce se soubory v lokálním repository.

Práce se soubory v lokálním repository

1. Vytvořte prázdný lokální repositář.
2. Vytvořte soubory a.txt, b.txt, c.txt.
3. Proveďte commit souborů a.txt a b.txt.
4. Změňte soubory b.txt a c.txt a proveďte commit změn.
5. Vytvořte soubor d.txt, který bude ignorovaný.
6. Odstraňte soubor a.txt a proveďte commit změny.

Zobrazení historie commitů

```
git log
```

Omezení počtu položek:

```
git log -5
```

Historie se statistikami

```
git log --stat
```

Formát historie

```
git log --pretty=oneline
```

```
panda@panda-mac nette $ git log -5 --pretty=oneline
02fb1cd7a7600e2c8cd75951f1fbd83fd0da89b8 DI: added possibility to use non-FQN names in @inject annotations
b97472ded40f61834ddc324c2b98fbff0c8a703f Reflection: Added caching of parsed files to AnnotationsParser
215e2ebdb38f259ff860d35cdfdc775b7c659691 Merge pull request #1431 from JanTvrdik/debugger_https_nginx
ff63a8577caaae2b9869c9d8b41168812149f77c Debugger: fixed HTTPS detection on nginx
376fa942b23593d46e62006da8f83c71be5f987d Merge pull request #1428 from JanTvrdik/simple_loader
```

Formát historie

```
git log --pretty=short
```

```
panda@panda-mac nette $ git log -1 --pretty=short
commit 02fb1cd7a7600e2c8cd75951f1fbd83fd0da89b8
Author: Tomáš Blatný <blatny.tomas@seznam.cz>

    DI: added possibility to use non-FQN names in @inject annotations
```

Formát historie

```
git log --pretty=full
```

```
panda@panda-mac nette $ git log -1 --pretty=full
commit 02fb1cd7a7600e2c8cd75951f1fbd83fd0da89b8
Author: Tomáš Blatný <blatny.tomas@seznam.cz>
Commit: David Grudl <david@grudl.com>

    DI: added possibility to use non-FQN names in @inject annotations
```

Formát historie

```
git log --pretty=fuller
```

```
panda@panda-mac nette $ git log -1 --pretty=fuller
commit 02fb1cd7a7600e2c8cd75951f1fbd83fd0da89b8
Author:     Tomáš Blatný <blatny.tomas@seznam.cz>
AuthorDate: Sat Mar 1 14:41:17 2014 +0100
Commit:     David Grudl <david@grudl.com>
CommitDate: Mon Mar 3 17:01:24 2014 +0100

    DI: added possibility to use non-FQN names in @inject annotations
```

Graf historie

git log --graph

```
panda@panda-mac nette $ git log --graph --oneline
* 02fb1cd DI: added possibility to use non-FQN names in @inject annotations
* b97472d Reflection: Added caching of parsed files to AnnotationsParser
*   215e2eb Merge pull request #1431 from JanTvrdik/debugger_https_nginx
|\
| * ff63a85 Debugger: fixed HTTPS detection on nginx
|/
*   376fa94 Merge pull request #1428 from JanTvrdik/simple_loader
|\
| * 69c8d94 NetteLoader: removed redundant slash from $list property [BC break]
| * 6e92c40 NetteLoader: removed dependency on Nette\Object, simplified loader.php
* |   48a506e Merge pull request #1422 from greeny/form-regexp-fix
|\ \
| |/
| * ee0a701 Forms: removed deprecated Form::REGEXP [Closes #1412]
|/
*   0f2bc0c Merge pull request #1411 from s4muel/patch-1
```

Hledání viníka

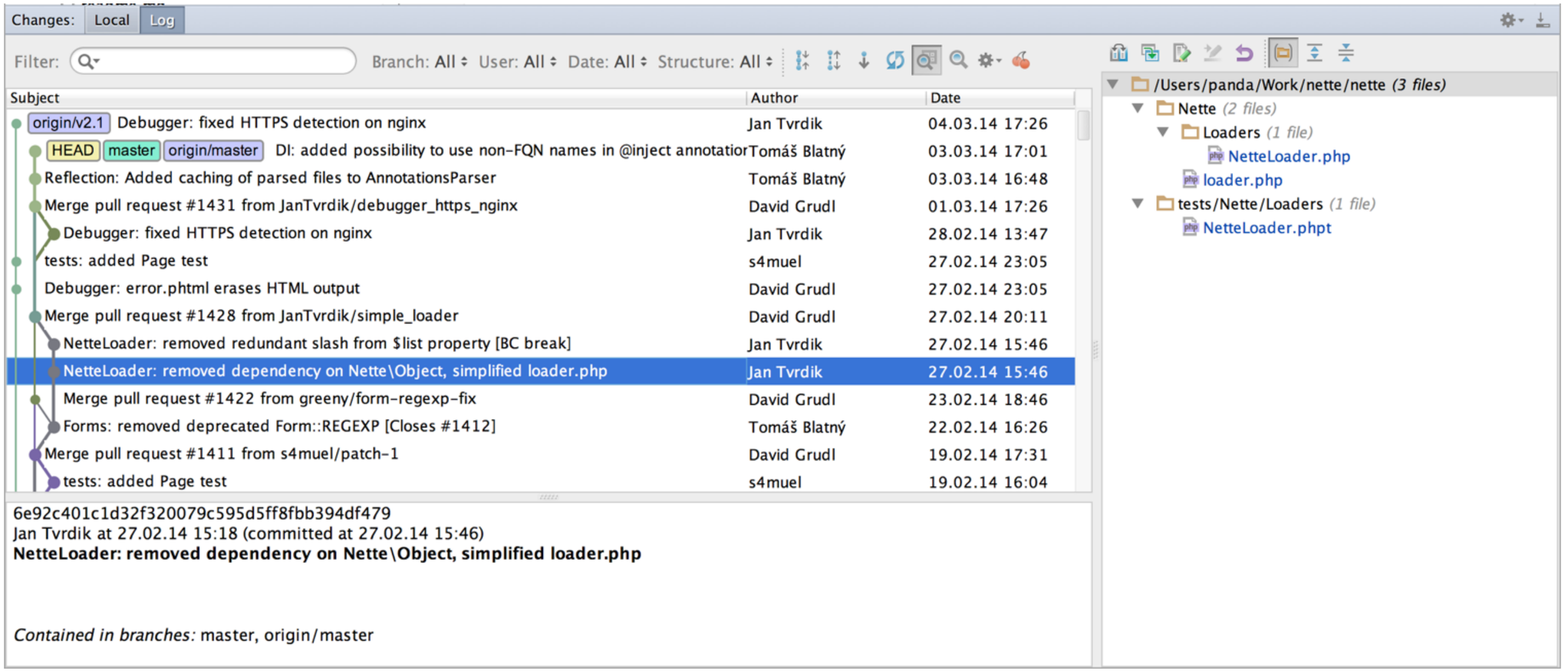
```
git blame <file>
```

Pro zadaný soubor najde, v jakém commitu, kým a kdy byly řádky naposledy modifikovány.

```
panda@panda-mac nette $ git blame Nette/Caching/Storages/IJournal.php
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200  1) <?php
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200  2)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200  3) /**
80eea36a Nette/Caching/ICacheJournal.php     (David Grudl 2011-02-02 00:38:23 +0100  4)  * This file is part of the Nette Framework (http://
nette.org)
bba3c4f2 Nette/Caching/Storages/IJournal.php (David Grudl 2012-01-02 04:27:15 +0100  5)  * Copyright (c) 2004 David Grudl (http://davidgrudl.com)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200  6)  */
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200  7)
16673791 Nette/Caching/Storages/IJournal.php (David Grudl 2011-04-13 03:36:29 +0200  8) namespace Nette\Caching\Storages;
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200  9)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 10) use Nette;
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 11)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 12)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 13) /**
8e003bc7 Nette/Caching/ICacheJournal.php     (David Grudl 2010-09-14 22:56:55 +0200 14)  * Cache journal provider.
13b21eed Nette/Caching/ICacheJournal.php     (David Grudl 2010-07-16 04:36:10 +0200 15)  *
13b21eed Nette/Caching/ICacheJournal.php     (David Grudl 2010-07-16 04:36:10 +0200 16)  * @author     Jan Smitka
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 17)  */
16673791 Nette/Caching/Storages/IJournal.php (David Grudl 2011-04-13 03:36:29 +0200 18) interface IJournal
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 19) {
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 20)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 21)         /**
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 22)          * Writes entry information into the journal.
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 23)          * @param  string $key
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 24)          * @param  array  $dependencies
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 25)          * @return void
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 26)          */
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 27)         function write($key, array $dependencies);
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 28)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 29)
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 30)         /**
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 31)          * Cleans entries from journal.
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 32)          * @param  array  $conditions
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 33)          * @return array of removed items or NULL when
performing a full cleanup
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 34)          */
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 35)         function clean(array $conditions);
bbbfb9b6 Nette/Caching/ICacheJournal.php     (Jan Smitka  2010-07-03 13:32:39 +0200 36)
6d1020e5 Nette/Caching/Storages/IJournal.php (David Grudl 2011-05-02 22:44:18 +0200 37) }
```

GUI

Na procházení historie často bývá nejlepší použít GUI.



Praktická část

Historie.

Historie

1. Vyzkoušejte různé formáty zobrazení historie.

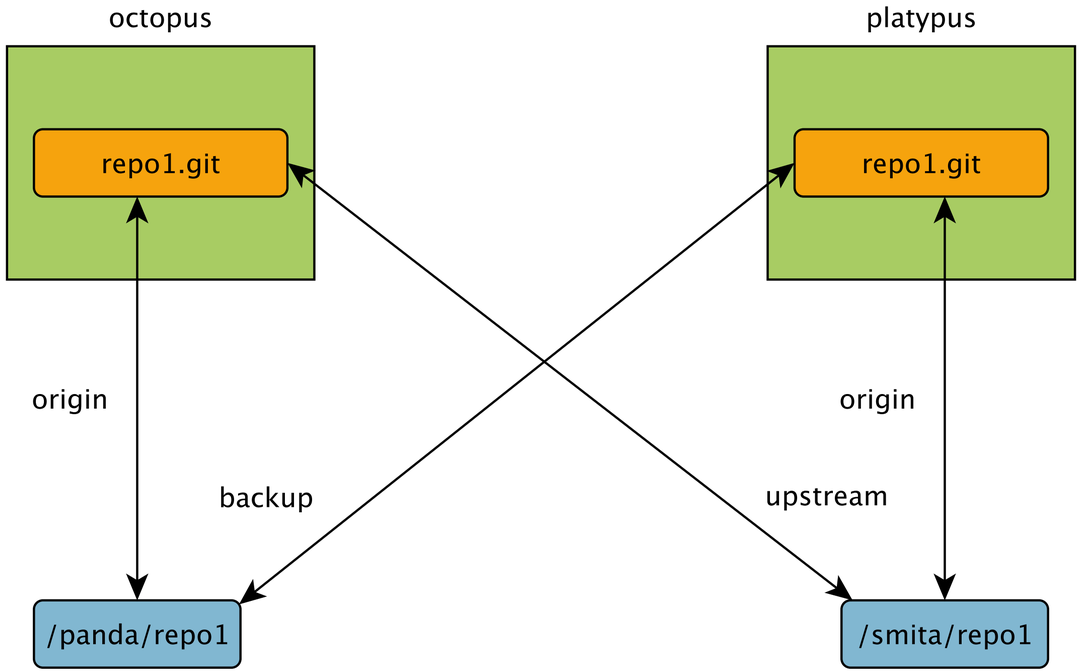
4.

Vzdálené repositáře.

Vzdálené repositáře M:N

Lokální repositář může mít více vzdálených repositářů, se kterými se synchronizuje.

Každý vzdálený repositář (remote) je pojmenovaný. Výchozí jméno je origin.



Vzdálené větve

Pro větve ve vzdálených repositářích se uchovává lokální kopie.

Pro větev master v remote origin se jmenuje origin/master – název možno používat při merge, zobrazení historie, ...

Synchronizace změn

Práce se soubory, operace commit, procházení historie apod. probíhá lokálně, není k nim potřeba připojení ke vzdálenému repository.

Změny se s nastavenými vzdálenými repositories synchronizují pomocí operací:

- **fetch** – stažení vzdálených změn, bez merge do lokální kopie.
- **pull** – stažení vzdálených změn a jejich merge do lokální kopie.
- **push** – nahrání změn.

Na serveru uloženo Bare Repository – repository bez Working Copy.

Komunikace s repositářem

S repository lze komunikovat pomocí několika protokolů:

- Souborový systém – lokální přístup k souborům.

  /mnt/remote/repository/repo.git nebo file:///mnt/remote/repository/repo.git

- HTTP(S) – read only, zápis vyžaduje WebDAV.

  http://server.lynt.cz/public/repo.git

- Git – protokol bez ověřování, vhodný pouze pro veřejné read-only repository.

  git://github.com/nette/nette.git

- SSH – zabezpečený R/W přístup.

  git@git.lynt.cz:repo.git nebo ssh://git@git.lynt.cz/repo.git

Zobrazení vzdálených repositářů

```
git remote
git remote show
```

```
panda@panda-mac nette $ git remote show
origin
```

```
git remote -v show
```

```
panda@panda-mac nette $ git remote -v show
origin git@github.com:nette/nette.git (fetch)
origin git@github.com:nette/nette.git (push)
```

Přidání vzdáleného repositáře

```
git remote add <name> <url>
```

```
panda@panda-mac ~ $ cd /Users/panda/Work/edu/git-1/repo/remote
panda@panda-mac remote $ git init --bare
panda@panda-mac test $ git remote add origin /Users/panda/Work/edu/git-1/repo/remote
```

Push

```
git push <remote> <branch>
```

Každá větev může mít Tracking Branch – vzdálenou větev, se kterou se bude synchronizovat. Nastavení pomocí

```
git push –u <remote> <branch>
```

Další push pak lze provést pomocí

```
git push
```

Push – podmínky

Pokud server má některé commity, které v lokálním repositáři nejsou, push je odmítnut. Je nutné provést stažení a merge změn pomocí pull.

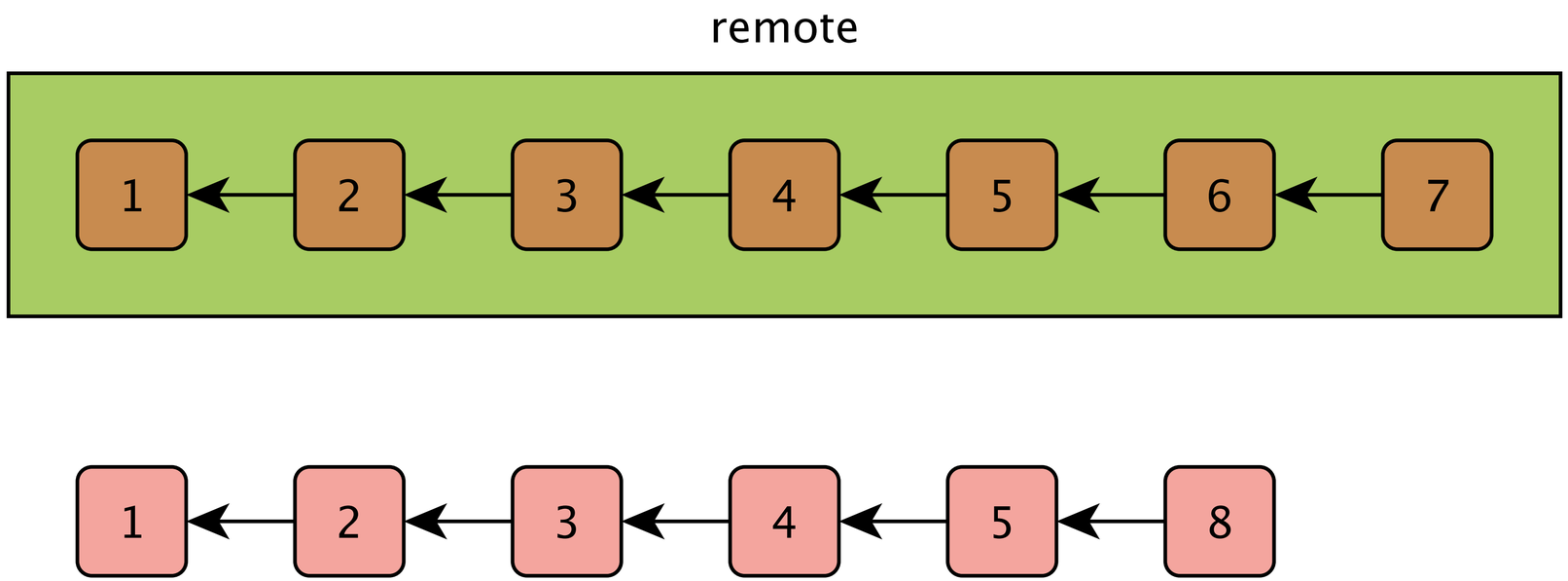
Pull

```
git pull <remote> <branch>
git pull
```

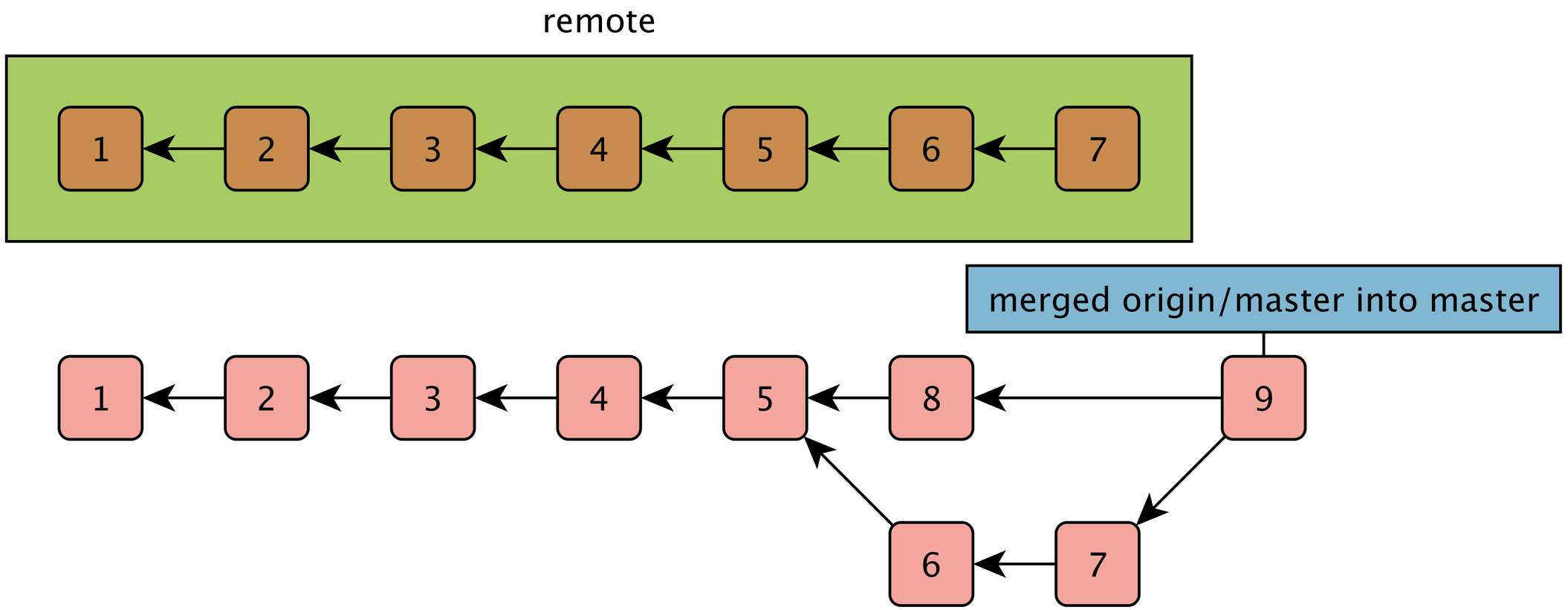
Zkrácená varianta funguje pouze s nastavenou Tracking Branch.

Proběhne stažení všech změn (fetch) a jejich merge do současné větve.

Před operací Pull



Po operaci Pull



Pull – podmínky

Pull se nezdaří, pokud by mělo dojít k přepsání souborů, které nejsou commitnuté.

```
panda@panda-mac test $ git pull
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /Users/panda/Work/edu/git-1/repo/remote
   46a5f3c..abfdf38  master     -> origin/master
Updating 46a5f3c..abfdf38
error: Your local changes to the following files would be overwritten by merge:
  INSTALL
Please, commit your changes or stash them before you can merge.
Aborting
```

Řešení: commit, nebo použití stash (bude probráno později).

```
panda@panda-mac test $ git log master
commit 46a5f3c6f680e4feda9b17e9b9439947efc34f5a
Author: Jan Smitka <jan@smitka.org>
Date:   Sun Jul 27 18:59:38 2014 +0200

    Test 1

commit 7fd3f2d3af630eab8c54e5135d95bd9a8fb72f9c
Author: Jan Smitka <jan@smitka.org>
Date:   Sun Jul 27 18:58:11 2014 +0200

    Initial commit.

panda@panda-mac test $ git log origin/master
commit abfdf38825d50555021e4a521b9bfc1da5ef782e
Author: Jan Smitka <jan@smitka.org>
Date:   Wed Aug 6 13:36:36 2014 +0200

    Test commit.

commit 46a5f3c6f680e4feda9b17e9b9439947efc34f5a
Author: Jan Smitka <jan@smitka.org>
Date:   Sun Jul 27 18:59:38 2014 +0200

    Test 1

commit 7fd3f2d3af630eab8c54e5135d95bd9a8fb72f9c
Author: Jan Smitka <jan@smitka.org>
Date:   Sun Jul 27 18:58:11 2014 +0200

    Initial commit.
```

Pull – konflikty

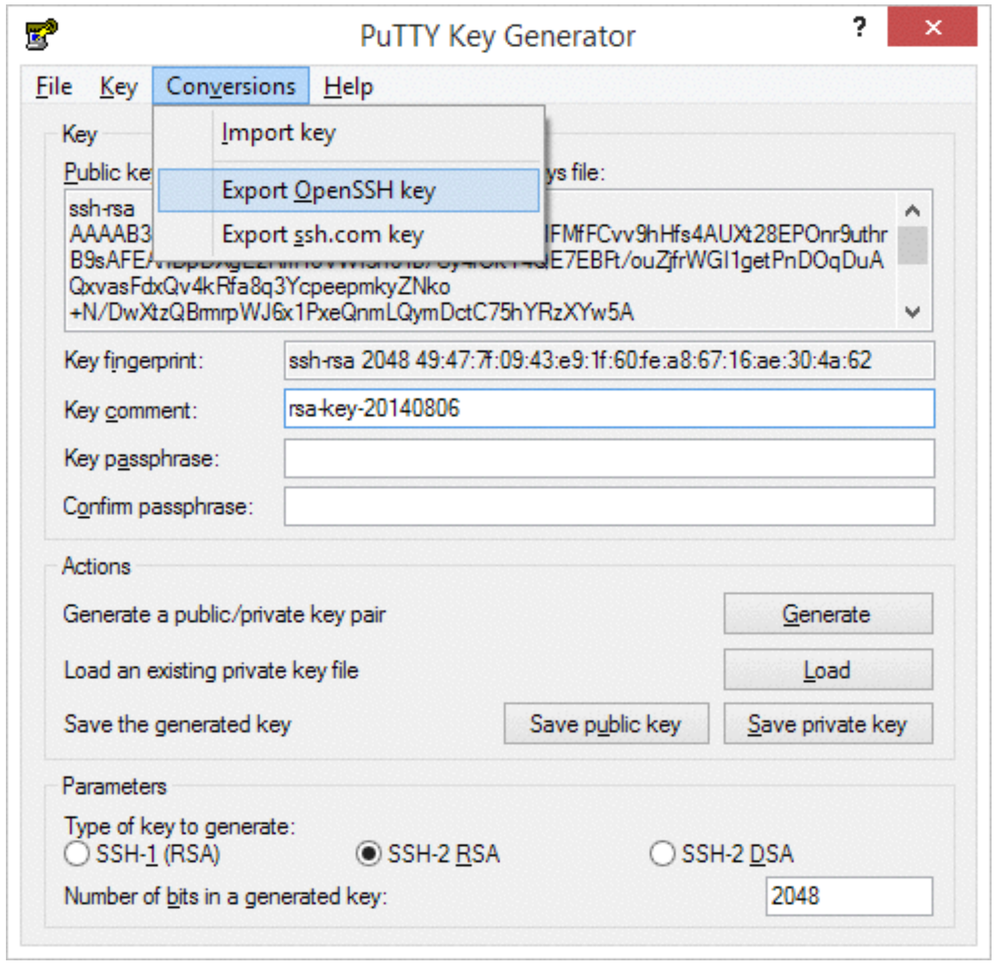
Při merge změn může dojít ke konfliktům.

Řešení stejné, jako u lokálních větví – bude probráno později.

Praktická část

Nastavení SSH.

PuTTYgen


PuTTY Key Generator
File Key Conversions Help
Import key
Export OpenSSH key
Export ssh.com key
Key
Public ke
ys file:
ssh-rsa
AAAAB3
IFMfFCvv9hHfs4AUXt28EPOnr9uthr
B9sAFE
E7EBFt/ouZjfrWGI1getPnDOqDuA
QxvasFdxQv4kRfa8q3YcpeepmkyZNko
+N/DwXtzQBmrpWJ6x1PxeQnmLQymDctC75hYRzXYw5A
Key fingerprint:
ssh-rsa 2048 49:47:7f:09:43:e9:1f:60:fe:a8:67:16:ae:30:4a:62
Key comment:
rsa-key-20140806
Key passphrase:
Confirm passphrase:
Actions
Generate a public/private key pair
Generate
Load an existing private key file
Load
Save the generated key
Save public key
Save private key
Parameters
Type of key to generate:
SSH-1 (RSA)
SSH-2 RSA
SSH-2 DSA
Number of bits in a generated key:
2048

~/.ssh/config

```
Host git.lynt.cz
      HostName octopus.lynt.cz
      Port 22
      IdentityFile ~/.ssh/certificate.pem
      User git
```

Praktická část

Práce se vzdálenými repositáři.

Práce se vzdálenými repositáři.

1. Naklonujte repositář git.lynt.cz:edu/test-1.git

A

2. Vytvořte soubor sum.php s funkcí sum($a, $b), která sečte zadaná čísla.

B

2. Vytvořte soubor price.php s funkcí price($unitPrice, $qty), která vypočte cenu zadaného počtu položek.

C

Vytvořte soubor output.php s funkcí print_item($text, $price), která vypíše zadaný název položky a její cenu na samostatný řádek.

3. Proveďte commit a push.
4. Proveďte pull a stáhněte změny ostatních.

5.

Správa tagů.

Typy tagů

Git rozlišuje následující typy tagů:

- **Lightweight** – pouze ukazatel na specifický commit.
- **Annotated** – uložen jako samostatný objekt s ukazatelem – má checksum, autora, zprávu, datum vytvoření, ...

Seznam tagů

```
git tag
git tag -l
```

```
panda@panda-mac nette $ git tag -l
v0.7
v0.8
v0.9.0
v0.9.1
v0.9.2
v0.9.3
v0.9.4
v0.9.5
v0.9.6
v0.9.7
v2.0.0
...
```

Zobrazení detailů zadaného tagu

```
git show <name>
```

```
panda@panda-mac nette $ git show v2.0.0
commit 013c8ee92624e21f45afd6b23f64997fd7285e50
Author: David Grudl <david@grudl.com>
Date:   Thu Feb 2 02:40:05 2012 +0100

    Released version 2.0

diff --git a/Nette/common/Framework.php b/Nette/common/Framework.php
index e4167fc..7c2c722 100644
--- a/Nette/common/Framework.php
+++ b/Nette/common/Framework.php
@@ -25,7 +25,7 @@ final class Framework

        /** Nette Framework version identification */
        const NAME = 'Nette Framework',
-               VERSION = '2.0-beta',
+               VERSION = '2.0',
                REVISION = '$WCREV$ released on $WCDATE$';

        /** @var bool set to TRUE if your host has disabled function ini_set */
diff --git a/version.txt b/version.txt
index 5d2cc71..5343ea4 100644
--- a/version.txt
+++ b/version.txt
@@ -1 +1 @@
-Nette Framework 2.0-beta (revision $WCREV$ released on $WCDATE$)
+Nette Framework 2.0 (revision $WCREV$ released on $WCDATE$)
```

Vytvoření lightweight tagu

```
git tag <name> <commit>
```

Pokud je vynechán commit, použije se HEAD.

```
panda@panda-mac test $ git tag v1.0
panda@panda-mac test $ git tag v0.1 7fd3f2d
panda@panda-mac test $ git tag
v0.1
v1.0
```

Vytvoření anottated tagu

```
git tag -a <name> <commit>
git tag -a <name> -m <message> <commit>
```

Push a tagy

Při operaci Push nejsou tagy nahrány na server, nutno vynutit flagem:

```
git push ––tags
```

Při operaci Pull jsou tagy v repositáři staženy, není nutno přidávat žádné flagy.

Praktická část

Vytváření tagů.

Vytváření tagů.

1. Vytvořte tag ve tvaru <příjmení> na posledním commitu v repositáři.
2. Vytvořte tag ve tvaru <jméno> na commitu, který jste vytvářeli.
3. Proveďte push tagů do repositáře.

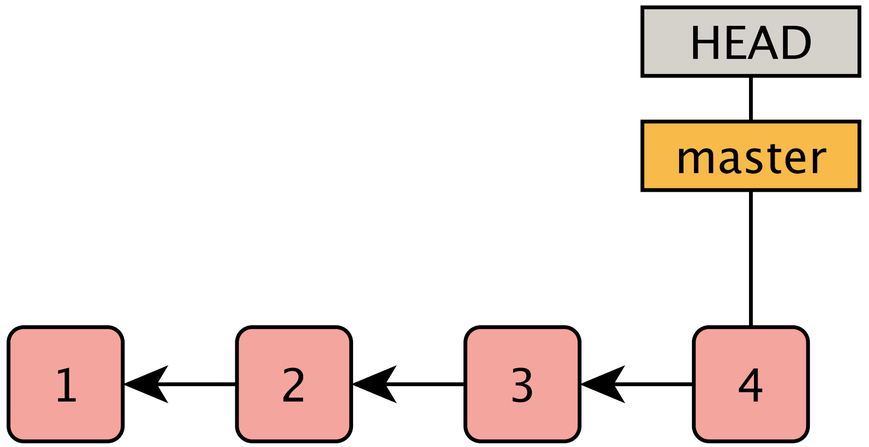
6.

Využití větví.

Větve v Gitu

Větve jsou pouze ukazatele na commity – jejich vytváření je velmi rychlé.

Odkaz je vždy na poslední commit dané větve.

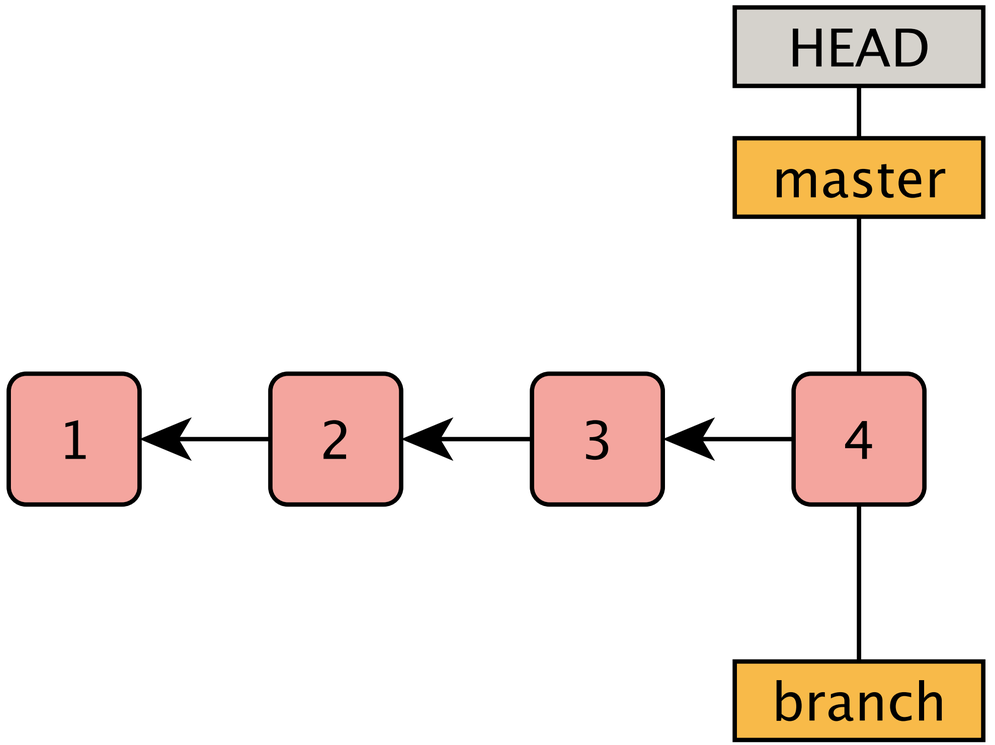


Vytvoření nové větve

```
git branch <branch>
```

Vytvoří novou větev, která bude mít vrchol na HEAD.

Vytvoření nové větve



Přepnutí větve

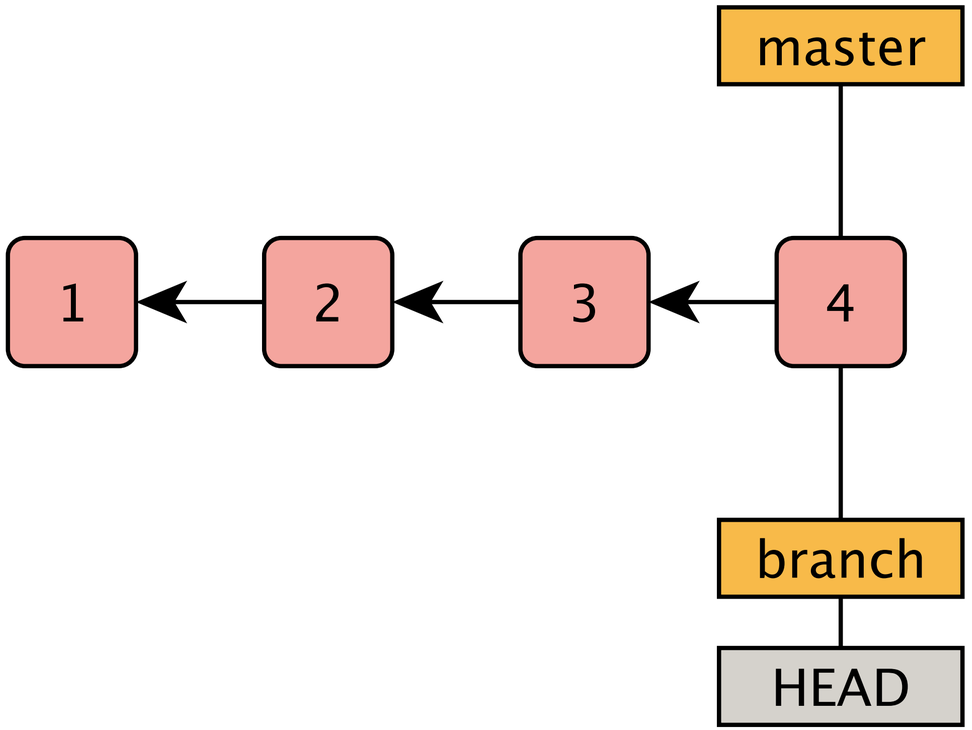
```
git checkout <branch>
```

Přepne HEAD na jinou větev.

Větev není možné přepnout, pokud by došlo k přepsání některých rozpracovaných změn (modifikované Tracked soubory a soubory ve Staging Area):

```
panda@panda-mac conflict $ git checkout master
error: Your local changes to the following files would be overwritten by checkout:
  index.htm
Please, commit your changes or stash them before you can switch branches.
Aborting
```

Vytvoření a přepnutí větve



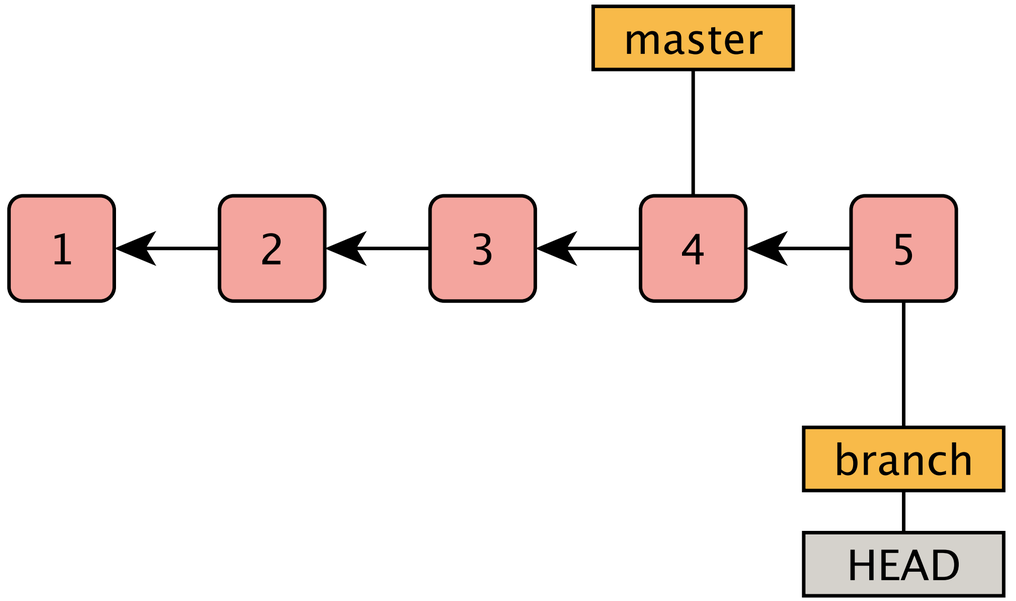
Vytvoření a přepnutí větve

```
git checkout –b <branch>
```

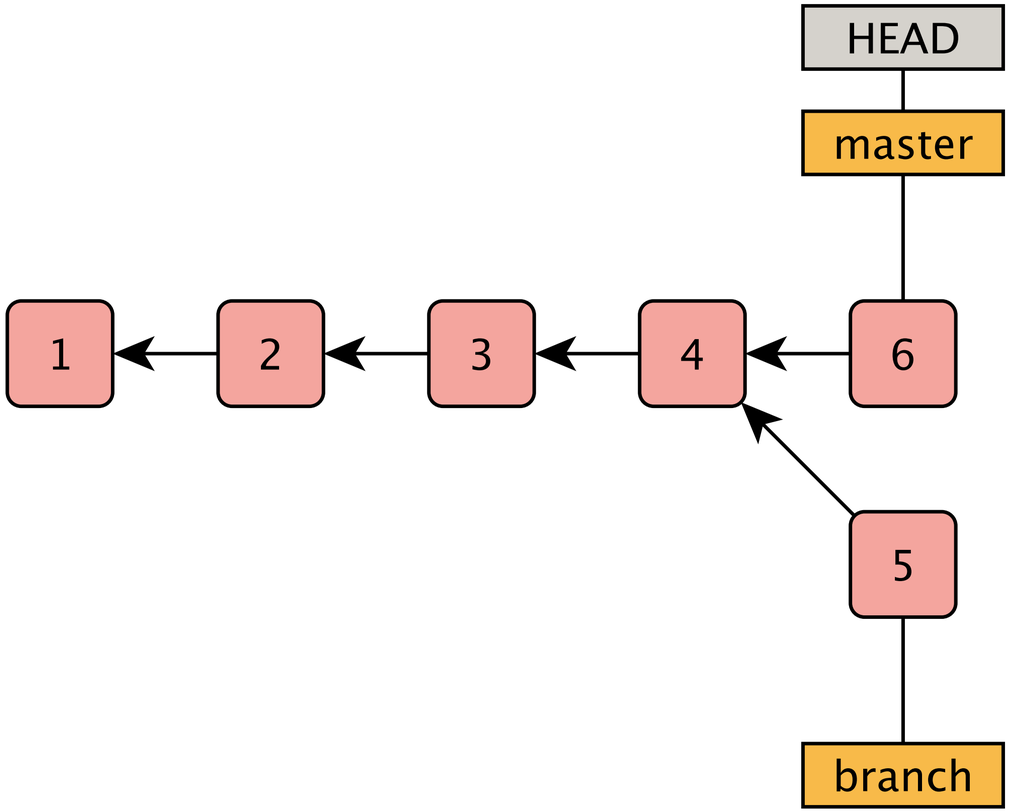
Zkratka pro:

```
git branch <branch>
git checkout <branch>
```

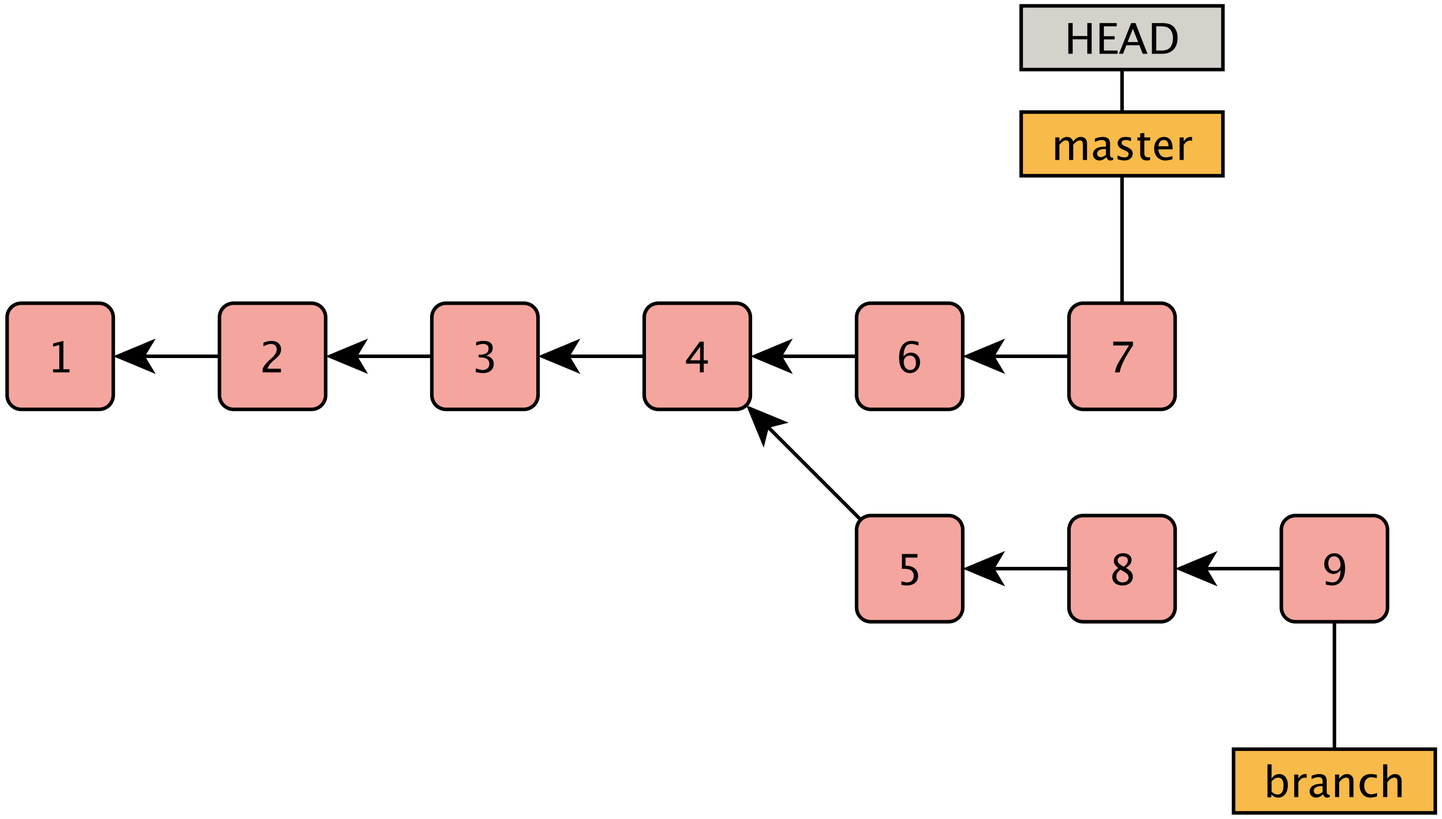
Izolovaný vývoj 1



Izolovaný vývoj 2



Izolovaný vývoj 3



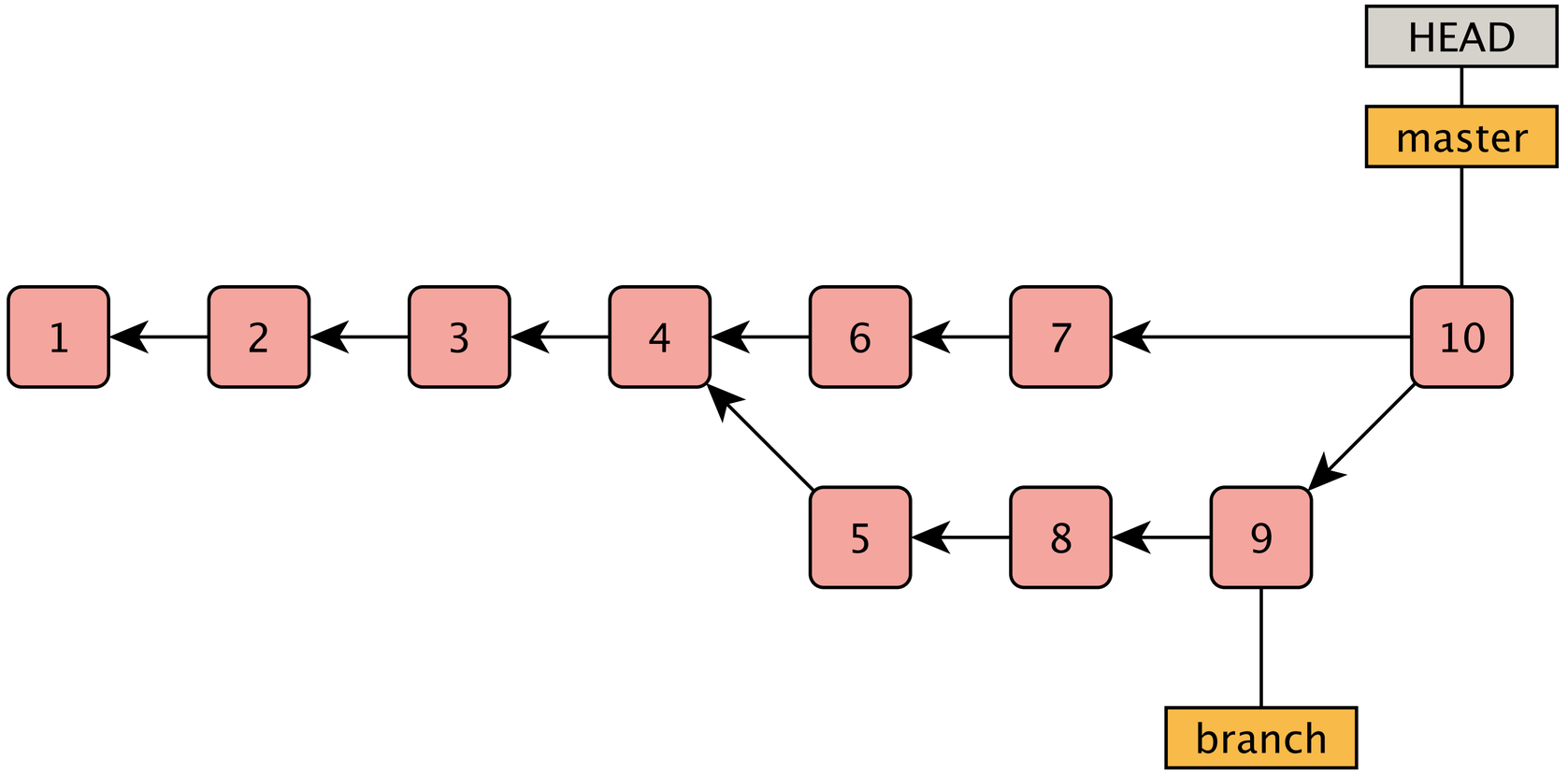
Merge větví

```
git merge <branch>
```

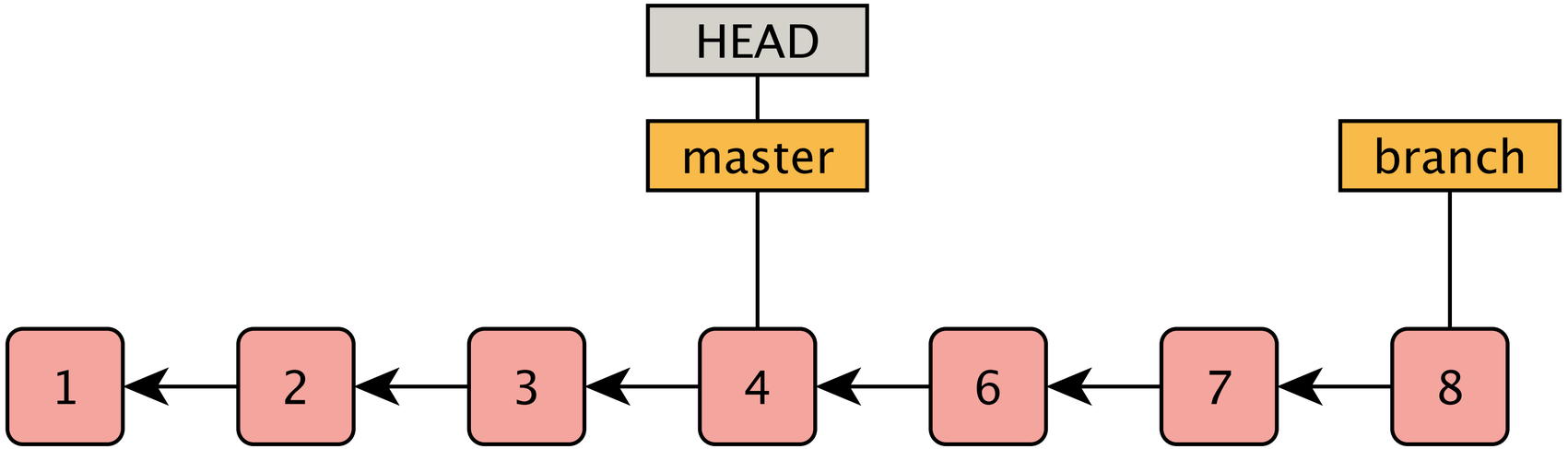
Provede Merge větve **<branch>** do **aktuální větve**. Merge může probíhat několika strategiemi, nejdůležitější:

- **Recursive Merge** – provede rekurzivní vyhledání společného předka a změny postupně slučí. Vytvoří nový commit.
- **Fast Forward** – pouze přesune ukazatel na větev dopředu v historii. Použitelné pouze v případě, že společný předek obou větví je vrchol aktuální větve.

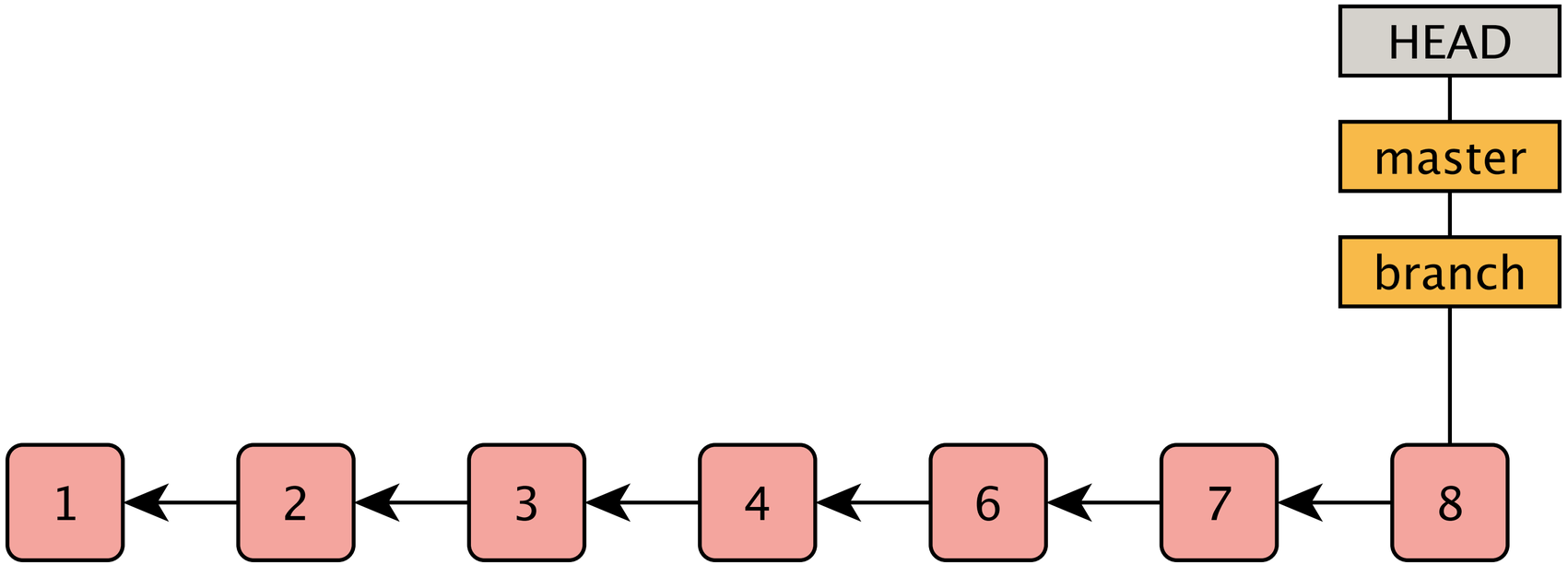
Recursive Merge



Fast Forward Merge 1



Fast Forward Merge 2



Strategie Merge

Pokud je to možné, je použita strategie Fast Forward. Možno zabránit, pokud se použije flag --no-ff:

```
git merge ––no–ff <branch>
```

Odstranění větve

```
git branch –d <branch>
```

Odstraní větev zadaného jména. Nelze odstranit větev, která doposud nebyla Merged do aktuální větve. Nutno použít přepínač -D:

```
git branch –D <branch>
```

Zobrazení větví, které byly Merged

```
git branch --merged
```

Zobrazí větve, které již byly Merged do aktuální větve, tj. ty, které jsou již plně začleněny.

Zobrazení větví, které nejsou Merged

```
git branch --no-merged
```

Zobrazení větví, které doposud nebyly Merged.

Praktická část

Vytváření a správa větví.

Vytváření a správa větví

A

1. Větev f-overpriced.
2. Pokud je suma dvou čísel vyšší, než 1000, bude naúčtováno o 5% víc.

B

1. Větev f-extra-beer.
2. Pokud má zákazník na účtu více než 10 položek jednoho typu, bude platit jednu navíc.

C

1. Větev f-nice-output
2. Zkrášlete výstup tak, aby obsahoval i počet jednotek (další parametr) a formát částky byl dle českých konvencí.

3. Proveďte push větve do repositáře.

Konflikty

Při Merge větví může docházet ke konfliktům – merge se nedokončí, konfliktní soubory jsou označeny v git status a obsahují data z obou commitů.

```
panda@panda-mac conflict $ git status
On branch master
You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

  both modified:   index.htm

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

Konflikt v souboru

```
panda@panda-mac conflict $ cat index.htm
<<<<<<< HEAD
<p>Nějaký Pokus</p>
=======
<p>První pokus.</p>
>>>>>>> merged-branch
```

V horní části jsou změny z aktuální větve, v dolní změny, které měly být sloučeny.

Soubor je nutné opravit a odstranit značky označující konflikt.

Vyřešení konfliktu

Po opravě souboru je nutné označit, že byl konflikt v něm vyřešen:

```
git add <file>
```

Merge se po opravě všech souborů dokončí commitem:

```
git commit
```

Commit Message bude obsahovat informace o konfliktu.

Praktická část

Řešení konfliktů.

Řešení konfliktů.

1. Proveďte merge svojí větve do master a vyřešte případné konflikty.

7.

Práce se stash.

Stash v Git

Oblast, do které je možné dočasně uložit rozpracované změny bez toho, aby bylo nutné je commitovat.

Do Stash se uloží **modifikované Tracked soubory** a soubory ze **Staging Area**.

Vhodné při Merge a přepínání větví.

Uložení do Stash

```
git stash
git stash save
git stash save <message>
```

Pokud není zadána message, použije se hash commitu na HEAD a začátek commit message.

```
panda@panda-mac conflict $ git stash
Saved working directory and index state WIP on master: 50a3991 Merge branch 'issue'
HEAD is now at 50a3991 Merge branch 'issue'
```

Zobrazení obsahu Stash

```
git stash list
```

```
panda@panda-mac conflict $ git stash list
stash@{0}: WIP on master: 50a3991 Merge branch 'issue'
stash@{1}: WIP on issue: 0707cf9 Test
```

Aplikace změn ze Stash

```
git apply
git apply --index
```

Aplikuje změny, které byly jako poslední přidány do Stash.

Pokud je uveden flag --index, tak se obnoví i původní Staging Area, v opačném případě se změny ze Staging aplikují, ale nepřidají se do Stage.

```
git apply <stash>
```

Aplikuje zadanou změnu ze Stash, např.:

```
git apply stash@{2}
```

Odstranění změn ve Stash

Aplikované změny ve stash zůstanou, po aplikaci je vhodné je smazat:

```
git stash drop
git stash drop <stash>
```

Změny je možné aplikovat a smazat jedním příkazem:

```
git stash pop
git stash pop <stash>
```

Aplikace změn do nové větve

Změny je možné aplikovat do nové větve:

```
git stash branch <branch>
git stash branch <branch> <stash>
```

Zobrazení detailů změny

Zobrazení shrnutí změn

```
git stash show
git stash show <stash>
```

Zobrazení změn v souborech (diff)

```
git stash show -p
git stash show -p <stash>
```

Praktická část

Práce se Stash.

Práce se Stash.

1. Na větvi master přidejte komentář se jménem autora ke všem funkcím. Změny přidejte do Staging, ale necommitujte.
2. Přepněte se do svojí větve a přidejte phpDoc komentáře k funkci, kterou jste vytvářeli. Proveďte commit. Použijte stash, pokud to bude nutné.
3. Přepněte se zpět na master a proveďte commit změn provedených v kroku 1. Vyřešte případné konflikty.

8.

Vzdálené větve.

Vzdálené větve

Kopie vzdálených větví se ukládají do lokálního repositáře, nelze s nimi však pracovat.

Pojmenovány podle vzoru

```
<remote>/<branch>
```

Operace Pull

Při pull se nejprve aktualizuje kopie vzdálené větve a poté se provede Merge do současné větve.

Operace Push

Ve výchozím nastavení se vzdálená větev jmenuje stejně. Je možné uvést jiný název vzdálené větve:

```
git push <remote> <local-name>:<remote-name>
```

Provede push lokální větve local-name, ve vzdáleném repository se bude jmenovat remote-name.

Mazání vzdálených větví

```
git push <remote> :<remote-name>
```

Smaže vzdálenou větev remote-name – jedná se o push lokálního ničeho na vzdálený server pod zadaným názvem.

Praktická část

Operace se vzdálenými větvemi.

Operace se vzdálenými větvemi

1. Odstraňte ze vzdáleného repositáře svojí větev.

9.

Workflow.

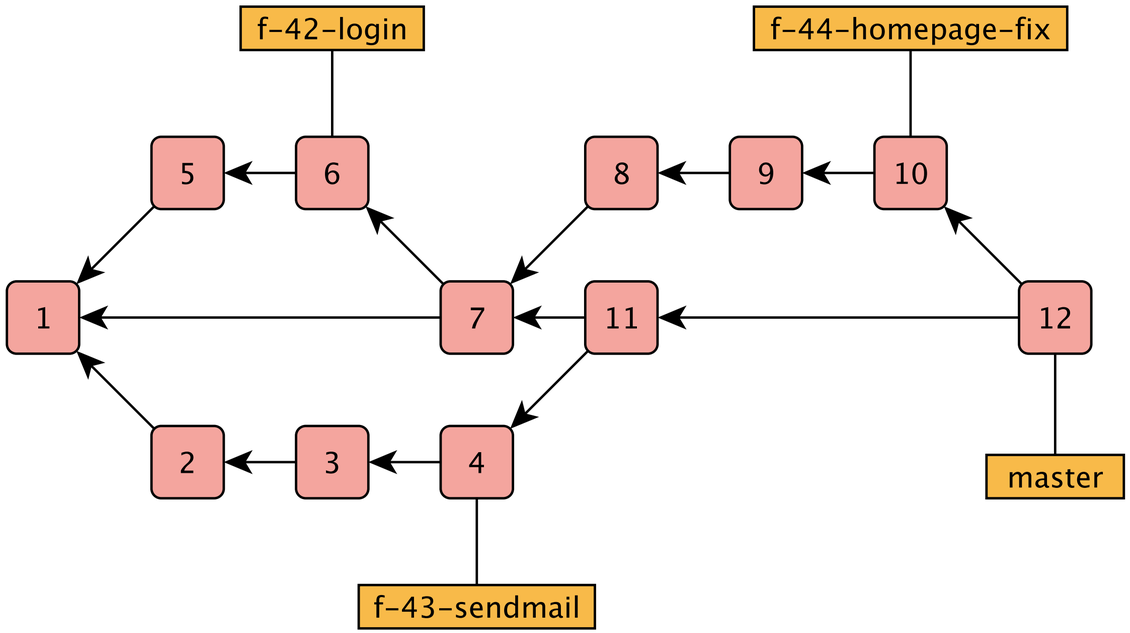
Workflow

Rychlé a levné větve v gitu umožňují celou řadu různých stylů práce, které bývají efektivnější než lineární vývoj. Základní vzory:

- **Feature (Topic) Branches**
- **Git Flow**
- **Github Flow**

Feature Branches

Vývoj neprobíhá přímo na hlavní větvi. Pro každou implementovanou feature se vytváří samostatná větev, vývoj probíhá nezávisle.



Feature Branches – pravidla

Jednotlivé větve by měly být jednotně pojmenovány podle určeného klíče – všichni vývojáři musí dodržovat.

Název Feature Branch musí být lokální.

Merge Feature Branches by měl být proveden s flagem --no-ff.

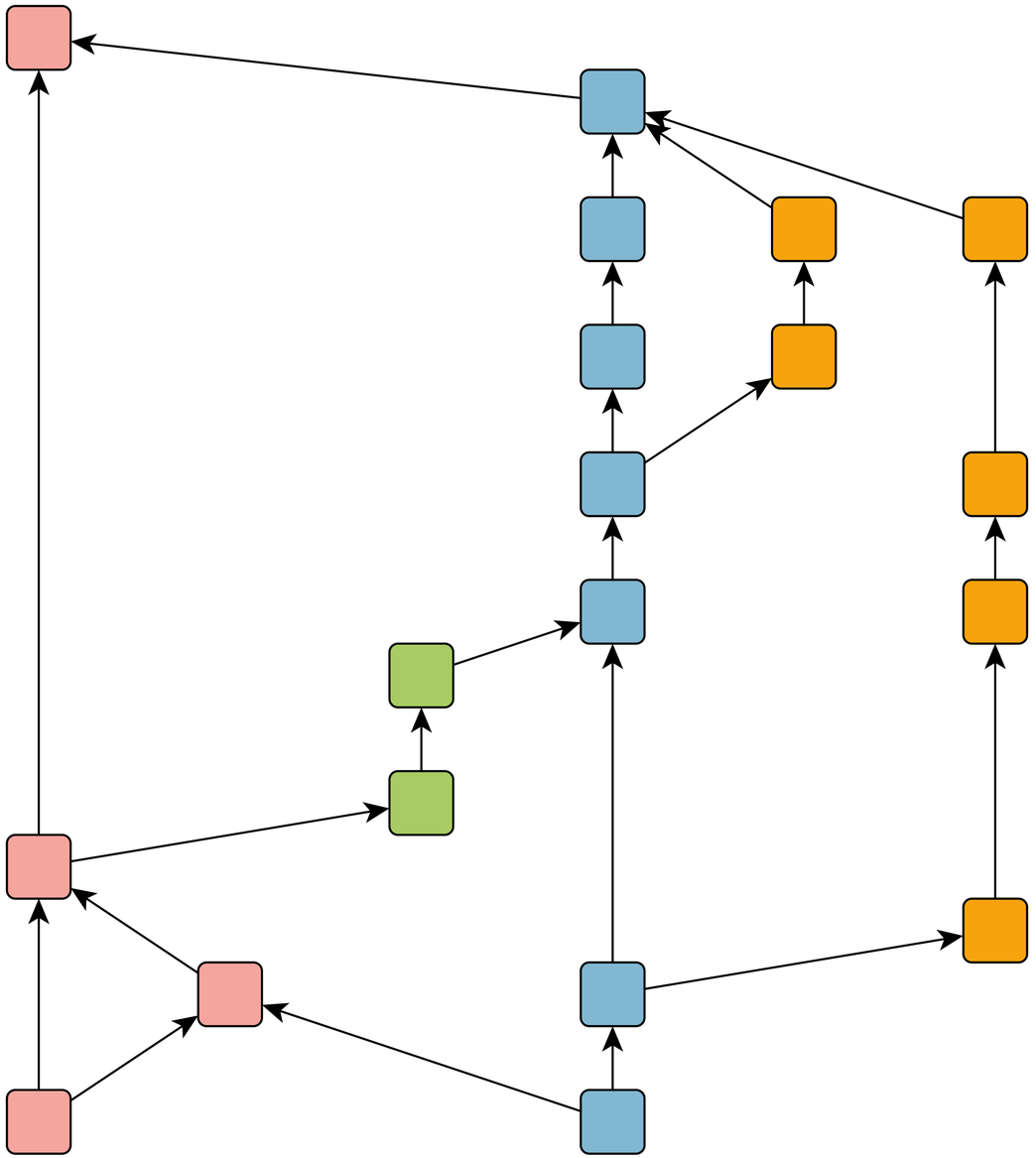
Pro každou změnu musí být vytvořena samostatná Feature Branch. Výjimku tvoří rychlé hotfixy, kde je režie na vytvoření větve větší, než změna samotná, a mohou být provedeny přímo na masteru.

Git Flow

Poměrně složité workflow, které používá celou řadu větví s různými stupni stability.

Vhodné pro dlouhodobé projekty s většími vydáními (releases).

master hotfix release develop feature-1 feature-2



Git Flow – větve

Master – použita jako hlavní větev projektu, do které se mergují publikované releases.

Hotfix – rychlé opravy, které je potřeba rychle dostat do releases.

Release – větve určené pro opravy v jednotlivých releases.

Develop – vývojová větev, na které probíhá vývoj dalších releases.

Feature – feature branches pro izolovaný vývoj funkcí.

Více na http://nvie.com/posts/a-successful-git-branching-model/

Github Flow

Zjednodušené workflow, které je určené pro projekty s častými inkrementálními releases, které jsou vyvíjeny na GitHubu.

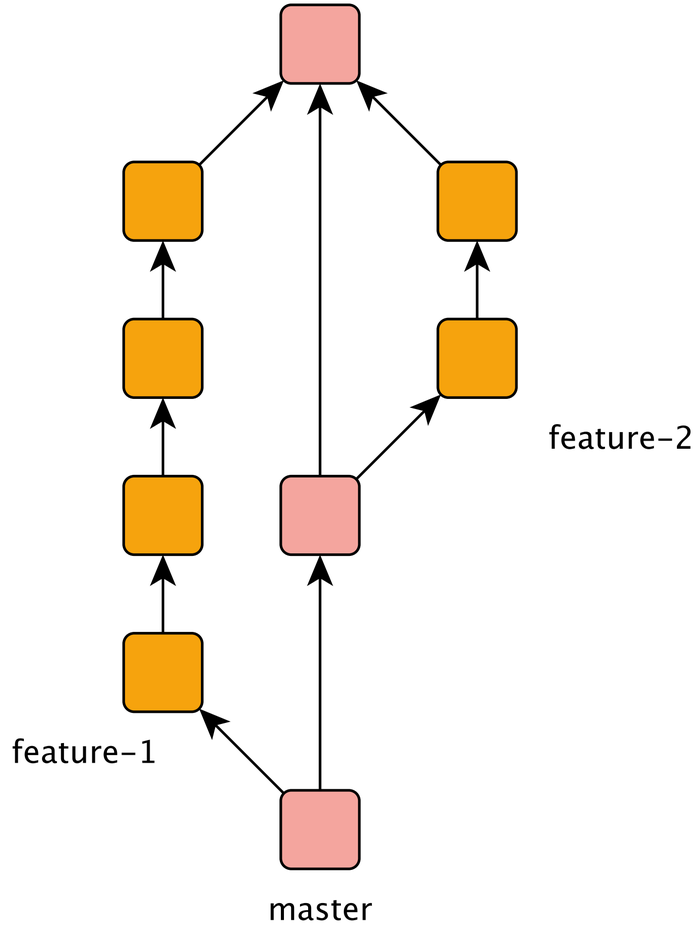
Využívá Feature Branches.

Github Flow – větve

Master je vždy stabilní a připraven k vydání.

Pro vývoj jsou použity **Feature Branches**.

Začlenění probíhá formou **Pull Requests**, kde probíhá review a diskuze o navrhovaných změnách.



10.

Patche.

Výpis rozdílů

Mezi Working Copy a Staging Area

```
git diff
```

Mezi Staging Area a daným commitem

```
git diff --staged [<commit>]
```

Mezi Working Copy a daným commitem

```
git diff <commit>
```

Mezi dvěma commity

```
git diff <commit> <commit>
git diff <commit>..<commit>
```

U všech variant je možné uvést cestu, které se má diff týkat.

```
panda@panda-mac conflict $ git diff --staged
diff --git a/index.htm b/index.htm
index 70c6176..6a3cd03 100644
--- a/index.htm
+++ b/index.htm
@@ -1 +1 @@
-<p>Nějaký První Pokus</p>
+<p>Nějaký Interní První Pokus</p>
```

Uložení patche

```
git diff ... > file.patch
```

Přesměruje výstup do file.patch.

Aplikace patche

Ověření, zda patch jde aplikovat

```
git apply --check <file>
```

Aplikace patche

```
git apply <file>
```

Patche s commity

```
git format-patch -M <commit>
```

Pro každý commit mezi HEAD a zadaným commitem vytvoří jeden patch soubor, který je možné např. zaslat e-mailem.

Flag -M donutí git vyhledávat i přesuny souborů.

Vytvořené soubory mají trochu jiný formát, než patche vytvořené pomocí git diff – obsahují také informace o autorovi, commit message apod.

Aplikace patchů s commity

```
git am <file>
```

Aplikuje commity z daného souboru.

Probíhá interaktivně:

```
panda@panda-mac conflict $ git am 0001-Test.patch 0002-Test-2.patch
Applying: Test
error: patch failed: index.htm:1
error: index.htm: patch does not apply
Patch failed at 0001 Test
The copy of the patch that failed is found in:
   /Users/panda/Work/edu/git-1/repo/conflict/.git/rebase-apply/patch
When you have resolved this problem, run "git am --continue".
If you prefer to skip this patch, run "git am --skip" instead.
To restore the original branch and stop patching, run "git am --abort".
```

Praktická část

Vytváření a aplikace patchů.

11.

Referencování commitů.

Přímé reference na commit

<sha1> – commit s daným checksum.

HEAD – commit aktuální Working Copy.

<branch> – vrchol lokální větve.

<remote>/<branch> – vrchol vzdálené větve, která byla stažená.

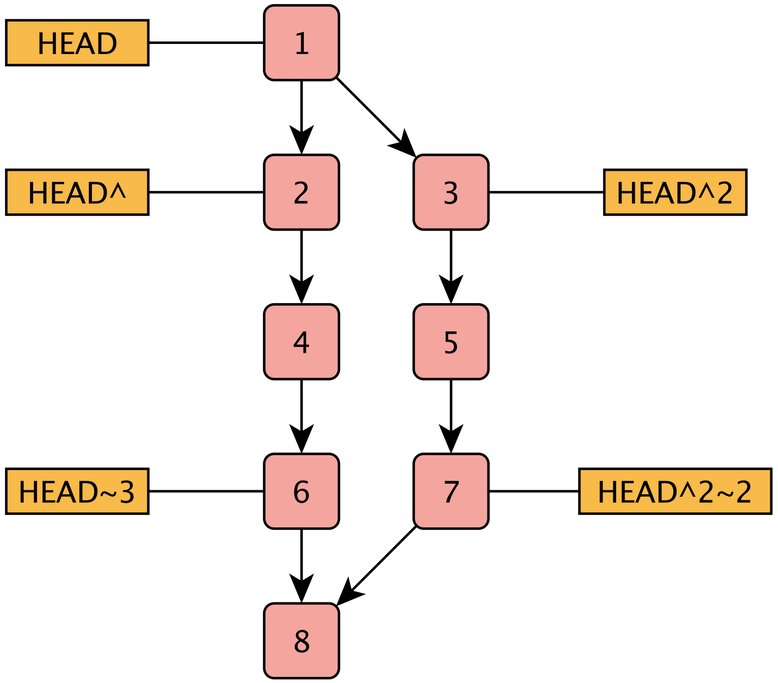
<tag> – commit označený tagem.

Relativní reference

<ref>^ – první rodič commitu <ref>, př.: HEAD^

<ref>^N – n-tý rodič commitu <ref> – dva rodiče mají např. merge commity, př.: HEAD^2

<ref>~N – N commitů zpět v historii (uvažuje prvního rodiče), př.: HEAD~3

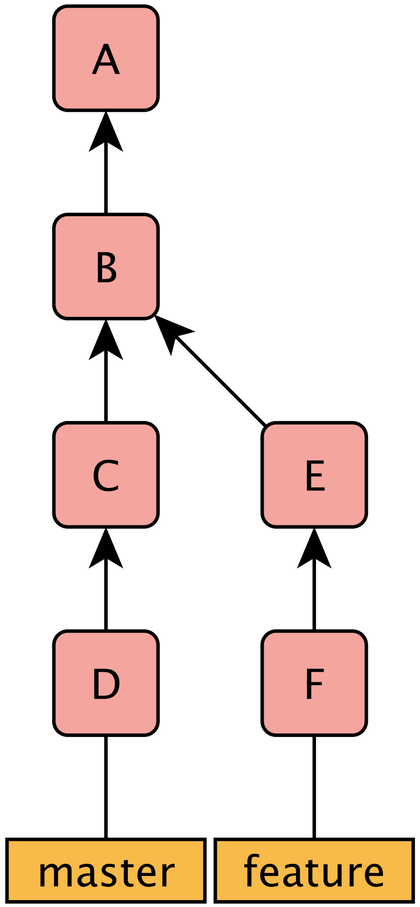


Reference na rozdíly stromů 1

<ref1>..<ref2> – double-dot, commity, ke kterým se lze dostat z ref2, ale ne z ref1:

master..feature: commity F, E

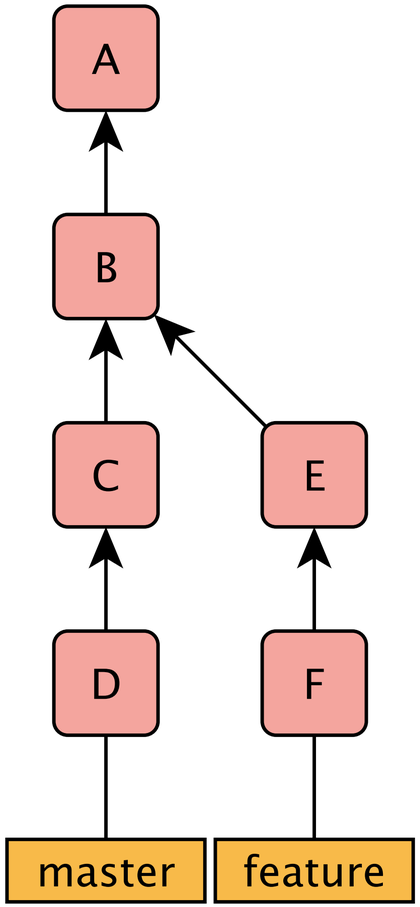
feature..master: commity D, C



Reference na rozdíly stromů 2

<ref1>...<ref2> – triple-dot, commity, ke kterým se lze dostat z ref2, nebo z ref1, ale ne z obou zároveň:

master...feature: commity F, E, D, C



12.

Závěr.

Další zdroje

Kniha Pro Git

http://git-scm.com/book

http://www.root.cz/knihy/pro-git/

Man Pages

http://git-scm.com/docs

Interaktivní cheatsheet

http://ndpsoftware.com/git-cheatsheet.html

Otázky?