Source Code Management (SCM) standard

Lundegaard Git standard aneb Standard definující pravidla vytváření, distribuce, revize a deploymentu zdrojového kódu za použití nástroje Git SCM, konvence Git flow a hostingových služeb pro Git repositáře ve společnosti Lundegaard a dalších na projektech spolupracujících organizacích.

Účastníci (actors)

- *Hosting platform* je webová služba hostující Git repositáře a podporující kolaboraci vývojářských týmů – může být poskytována buď třetí stranou (Bitbucket.org, Github.com – tzv. on-demand služby) nebo provozována samotnou organizací (tzv. on-premise systém); pokud není stanoveno jinak, předpokládá se v standardně využití on-demand platformy Bitbucket.org nebo on-premise systému Atlassian Stash
- *Organization* je společnost aplikující tento standard (defaultně Lundegaard, spol. s r.o.) a zároveň agregační entita vytvořená na hostingové platformě sdružující hlavní repositáře projektů, jejichž kód vlastní nebo pro své klienty spravuje dané organizace.
- *Head developer (hlavní vývojář)* je pracovník organizace zodpovědný odborně za všechny vývojáře v organizaci. Určuje a dohlíží na plnění technických standardů ve vývoji. Hlavní vývojář Lundegaardu je také přímo zodpovědný za udržování tohoto standardu.
- *Platform administrator (správce platformy)* je pověřený pracovník organizace, který má k hostingové platformě administrátorská oprávnění a může tedy v ní vytvářet repositáře a přidělovat dalším uživatelům platformy k nim oprávnění; je přímo zodpovědný za zabezpečení přístupu ke zdrojovým kódům (tedy umožnění přístupu pouze členům projektového týmu případně dalším vývojářům organizace dle požadavků ředitele vývoje a v souladu s bezpečnostním režimem stanoveným pro daný projekt). Správcem platformy je defaultně hlavní vývojář, který tuto roli může dále delegovat na některé další vývojáře.
- *Main repository* je repositář vytvořený pro projekt na hostingové platformě v rámci organizace. Je primárním zdrojem zdrojového kódu projektu pro vývojáře i provoz (deployment).
- *Fork repository* je klon main repository, který si vytvoří vývojář na hostingové platformě za účelem propagace změn do main repository mechanismem pull requestu.
- *Local repository* je klon, který si vytvoří vývojář pro potřeby vlastního aktivního vývoje (working copy slouží pro vytváření zdrojového kódu projektu). Local repository se může nacházet bud v lokálním (pracovní stanice) nebo sdíleném vývojovém prostředí (vývojový server).
- *Deployment tool* je aplikace umožňující nasazovat automatizovaně projekt do konkrétních prostředí za využití větví v main repository a dle konfigurace definující konkrétní cílová běhová prostředí (*target runtime*) pro daný typ prostředí (*environment*).
- *Runtime* je běhové prostředí (server nebo aplikační cloud nakonfigurovaný v souladu s běhovými požadavky projektu-aplikace)
- *Environment* je prostředí pro běh projektu – rozlišujeme prostředí

    - *develop* - vývojové
    - *test* – tetovací (preview, UAT)
    - *production* – produkční (ostrý provoz)

Životní cyklus projektu

1. Obecná pravidla pro větvení verzování projektu

Všechny projekty využívají standardizované workflow pro větvení zdrojového kódu projektu v návaznosti na životní cyklus projektu Git flow popsané v dokumentech

- http://nvie.com/posts/a-successful-git-branching-model/
- https://github.com/nvie/gitflow (README.md)

Pravidla pojmenovávání větví a tagů

1. **feature** a **hotfix** větve musí být pojmenovány ve tvaru trackerid-short-name, kde

    1. **trackerid** je je číslo tasku z trackeru ve tvaru 12345 (např. Mantis) nebo XY-12345 (např. JIRA); pro feature není nezbytně nutný task v trackeru (a pokud neexistuje, potom se ve jménu větve trackerid neuvádí), pro hotfix je však nutný VŽDY
    2. short-name je zkrácený popis větve (max. 10 slov, pouze lower case písmena a číslice, slova oddělená pomlčkou)
       Příklady:
       - feature/openid-implementation
       - hotfix/12345-kontaktni-formular-ukladani-dat-do-tabulky

2. **release** branch se pojmenovává vždy číslem verze generovaným ve tvaru ***x.y.z*** *(**MAJOR.MINOR.PATCH**)* v souladu se sémantickým verzováním (semver.org)

    1. ***x*** je major číslo verze (1 až n), které se zvyšuje v případě **zpětně nekompatibilních změn v API – významného** te chnického redesignu projektu (typicky změna architektury) - **zpětně nekompatibilních změn v API**; o zvýšení major čísla verze rozhoduje nebo jej schvaluje vždy hlavní vývojář.
    2. ***y*** je minor číslo verze (0 až n) odpovídající **releasu** obsahujícímu nově přidané funkcionalitě při **zachování zpětné kompatibility**; pokud není stanoveno jinak, s každým dalším releasem se zvyšuje o jedničku.
    3. ***z*** je číslo patche (0 až n) odpovídající **zpětně kompatibilnímu** hotfixu (bugfixu); pokud není stanoveno jinak, s každým dalším hotfixem se zvyšuje o jedničku.
3. Tag commitu ukončeného **releasu** mergnutého do masteru přejímá v souladu s gitflow název releasu tj. ***x.y(.z)***
4. Tag commitu ukončeného **hotfixu** mergnutého do masteru je oproti gitflow ve tvaru ***x.y.z***, obdobně jako u releasu viz výše.

    1. Pozn.: protože git flow hotfix finish příkaz neumožňuje zadat vlastní jméno tagu, je nutné jeho vytvoření potlačit přepínačem -n a následně jej vytvořit manuálně – vzor postupu:

        ```
        git flow hotfix finish -n 12345-some-hotfix
        git checkout master
        git tag 1.1.5
        ```

        V případě standardního režimu distribuce změn (viz. kapitola 4) je za vytvoření tagu na master větvi zodpovědný vývojář, který přijímá pull request z hotfix branche.

V případě pochybností o správném pojmenování větve rozhoduje hlavní vývojář.

Pravidla commitování

Vývojář commituje tak často, jak je to možné, prostřednictvím „atomických commitů". **Atomické commitování** slouží k udržení přehlednosti historie změn a umožňuje relativně snadné revertování nežádoucích změn. Atomickým commitem se rozumí **nejm enší možná úprava,** která se týká **jednoho issue** a odpovídá jedné změně v kódu např. přidání metody, přidání šablony apod. Vývojář naopak nespojuje úpravy týkající se více issues do jednoho commitu.

Při mergování kódu **do větví určených k automatickému deploymentu** by vždy měl vzniknout nový commit (**merge musí mít přepínač --no-ff**), jinak nelze spolehlivě zajistit automatický deployment. Naopak v situaci, kdy vývojář provede merge dříve odevzdané větve develop do větve preview (master) a provede merge fast-forward, nevznikne nový commit a Bitbucket nepošle POST hook, který je nezbytný pro spuštění automatického deploymentu.

```
git checkout develop; git merge --no-ff feature/XYZ-123-issues-summary;
git checkout preview; git merge --no-ff develop;
```

Pravidla odevzdávání práce (pushování)

Vývojář odevzdává (pushuje) minimálně 1x denně, aby bylo možné v případě jeho nepřítomnosti předat práci jinému vývojáři a na jeho práci plynule navázat, testovat, provádět code review aj.

2. Vznik projektu

1. Vytvoření main repository (správce)

Správce platformy na základě pokynu projektového managera nebo na popud vývojáře projektu vytvoří main repository projektu na hostingové platformě. Repository by mělo mít název ve tvaru *client-solution* resp. *product*, *Client.Solution* případně *ClientSolution* resp. *Product* dle konvence aplikační technologie daného projektu, konkrétně

- *client-solution* resp. *produkt* pro platformy Java (J2SE/J2EE/Android) a PHP (nespecifické případy)
- *ClientSolution* resp. *Product* pro specifické případy PHP (např. Zend moduly), .NET

Případné spory o volbu správného názvu repository pro daný projekt rozhoduje hlavní vývojář.

Výsledná URL adresa vytvořeného main repository je ve tvaru https://*platform*/*vendor*/*project*.git, kde

- *platform* je doménová adresa hostingové plarformy (defaultně bitbucket.org, případně on-premise systém Stash)
- *vendor* je organizace, pod kterou main repository projektu patří (defaultně lundegaard)
- *project* je jméno repository projektu vytvořené dle výše popsané konvence

2. Inicializace repository (vývojář)

Po vytvoření main repository určený (případně v daný okamžik jediný) vývojář projektu provede iniciaci projektu a iniciaci git flow ve výchozím nastavení příkazy

```
mkdir project
cd project
git init
git flow init
git remote add upstream https://user@bitbucket.org/lundegaard/project.git
git push --set-upstream upstream develop
git push --all
```

3. Zapojení dalších vývojářů do projektu

Každý další vývojář, který se zapojí do projektu si vytvoři lokální klon a iniciuje git flow ve výchozím nastavení

```
git clone https://user@bitbucket.org/jprijmeni/project.git --origin
upstream
git flow init
```

4. Distribuce změn ve vývojovém týmu

V rámci distribuce změn, které provádí vývojáři v kódu projektu, rozlišujeme dva možné režimy: zjednodušený a standardní.

Ve zjednodušeném režimu není aplikováno code review machanismem pull requestu a není tutíž nezbytně potřeba, aby si vývojář vytvářel privátní kopii (fork) main repository. Všichni vývojáři na projektu mají v tomto režimu k main repository oprávnění write (push) a změny na existujících i nově vytvořených větvích pushují do něj (git push).

Ve standardním režimu nemají vývojáři s výjimkou těch, kteří mají za úkol provádět code review, oprávnění write (push) k větvím podléhajícím code review (standardně develop a master – dále reviewed branches). Hlavní vývojář stanoví, které větve podléhají code review a kdo jej bude zajišťovat, podle toho správce platformy nastaví vývojářům příslušná oprávnění read resp. write k main repository. Každý vývojář si v tomto režimu vytvoří na hostingové platformě privání fork main repository a p řidá si jej v nastavení lokálního klonu projektu jako origin remote:

```
git remote add origin https://user@bitbucket.org/user/project.git
```

Změny na větvích podléhajících code review pushuje vývojář do svého fork repository (git push origin), odkud je propaguje do main repository pull requesty následovně

1. z feature branche (feature/xxx) do develop
2. z release branche (release/x.y) do master
3. z hotfix branche (hotfix/nnnn-xxx) do master
4. V případě, že je pull request odmítnut, nebo komentován, vývojář dále zapracovává přípomínky do dané větve pushované do fork repository. Pokud je pull request přijat, může vývojář danou větev ze svého fork repository odstranit.

3. Deployment

Deployment se provádí mergováním změn připravených k nasazení do určitého cílového prostředí (*target environment*) do větve, která je pro toto prostředí určena, tj.

- *develop* do interního testovacího prostředí (prostředí kontinuální integrace, tzv. night builds) – (*internal testing environment)*
- *master* do produkčního prostredí (*production environment*)
- *release/x.y* do akceptačního (UAT) testovacího prostředí (*release testing environment*)
- *hotfix* do servisního testovacího prostředí (*maintenance testing environment*)

Z dané větve se změny nasadí do cílového prostředí buď ručně nebo automatizovaně.

Ruční deployment provádí vývojář nebo pracovník provozu v daném cílovém prostředí příkazem git pull na příslušné větvi pro dané prostředí viz. výše uvedená definice. Ruční deployment bude podporován pouze po dobu nezbytně nutnou, tedy do doby nahrazení automatizovaným deploymentem na daném projektu.

Automatizovaný deployment využívá konfigurovatelný CI tool (standardně Jenkins/Hudson), který umožňuje proces nasazení definovat ve formě opakovaně spustitelné úlohy. Úlohy se definují per environment (develop, test, production). Deploy úloha pro webovou aplikaci provádí typicky následující kroky

1. stáhne do svého workspace z Gitu obsluhou zadanou větev (testing environment) nebo tag (production environment)
2. Provede instalaci (PHP za využití composeru, Java za využití Mavenu nebo Antu)
3. Deployne výslednou instalaci do cílového běhového prostředí (PHP za využití rsync, Java přes ssh nebo k tomu určeným Maven goalem či Ant targetem)

Za konfiguraci úlohy automatizovaného deploymentu je trvale zodpovědný deployment manager, který ji provádí v součinnosti s vývojáři. Ten dle nastavení rolí v projektu přiděli v použitém CI systému oprávnění spouštět úlohu nasazení do daného cílového prostředí konkrétním dalším členům týmu.