Dokumentace k semestrální práci předmětu BI-GIT

Milan Vu

26. ledna 2019

Fáze 1 Slití repozitářů

1.1 Klon repozitáře

... nepotřebuje další komentář ...

1.2 Klon SVN repozitáře

... nepotřebuje další komentář ...

1.3 Struktura repozitářů a postup slití

Zjistil jsem, že se v 1. repozitáři nachází 28 commitů a ve 2. SVN repozitáři 29 commitů. Také jsem zaznamenal, že jsou poslední 2 commity 1. repozitáře, a poslední 3 commity 2. repozitáře odlišné. Pomohlo mi to pochopit, proč by se po slití mělo v repozitáři (resp. větvi master) nacházet 31 commitů.

Byly použity příkazy:

```
git rev-list --all --count
git remote add svn <svn repo path>
git fetch --all
git log --oneline --graph --all
```

Přepínač `-all` mi ukáže log všech commitů, na které ukazuje nějaká reference (`svn/master`) a ne jen `master` větev.

1.4 Rebase

Byl použit příkaz:

```
git log --oneline
```

pro zjištění adresy větve `master` a:

```
git rebase --onto svn/master 6493a7a master
```

který přesune větev `master` za větev `svn/master`. Poté jsem opravil konflikty nejdříve v souboru `index.html` a poté i v `js/edgebox.js`. Na závěr jsem pomocí:

```
git rebase --continue
```

dokončil rebase.

1.5 Smazání reference na SVN

Byl použit příkaz:

```
git remote remove svn
```

1.6 Návrh `.gitignore`

Byl vytvořen nový soubor `.gitignore`:

```
echo␣"*~"␣>>␣.gitignore
```

Tento soubor zajišťuje, že bude git ignorovat nové soubory, jejichž jméno končí vlnovkou a které se nachází kdekoliv v repositáři. Bude fungovat pouze v těch revizích, kde bude tento commit jejich rodičem.

1.7 Commit pod jiným uživatelem

Byly použity příkazy:

```
git add .gitignore
git commit --author="Petr <petr@edgebox.net>"
```

Přepínač `-author` je mi znám z přednášek.

1.8 Pročištění historie od ignorovaných souborů

Byl použit příkaz:

```
git filter-branch --force --tree-filter 'git rm -rf --ignore-unmatch *~ 'HEAD
```

který projde všechny commity a pro každý commit smaže pomocné soubory. Po třetím volání už `refs/heads/master` zůstaly nezměneny, tudíž v žádném commitu už zaručeně nebyly pomocné soubory.

Příkaz jsem našel na stackoverflow.

1.9 Pročištění referencí

Protože jsem volal minulý příkaz s atributem `-force`, `original/*` reference byly vyčištěny.

1.10 Odevzdání fáze 1

Byly použity příkazy:

```
git checkout -b phase1
git push --set-upstream --force origin phase1
```

Přepínač `-force` proto, protože to nebyl první pokus.

Fáze 2 Čištění repozitáře

2.1 FIX of FIX

```
git checkout master
git log --oneline
git rebase -i 8ae3296
```

Nejdříve jsem se vrátil na větev `master` pomocí `git checkout master`. Poté jsem pomocí `git log -oneline` vypsal historii commitů a udělal interaktivní rebase nad commitem, který byl před prvním FIX commitem. Následně jsem změnil pick na squash u celkem 6 commitů.

2.2 Windows konce řádků

Byly použity příkazy:

```
git filter-branch --force --tree-filter '
sed -i "" 's/^M$//g' index.html'
--tag-name-filter cat -- --all HEAD
```

Sed příkaz jsem našel na stackoverflow. Pak už jsem jen dosadil script do kanónu k smazání znaků ze všech commitů.

Pozn.: Z nějakého důvodu se změnily commity i před prvním výskytem CRLF a tak jsem musel použít přepínač pro zachování tagů.

2.3 Odstranění řádků „git-svn-id“ z commit messages

Byly použity příkazy:

```
git filter-branch -f --msg-filter '
sed -e "/git-svn-id:/d" -e "/Former-commit-id:/d"'
--tag-name-filter cat -- --all
```

Opět jsem se inspiroval na stackoverflow, ale přidal jsem tam tagy, aby se zachovaly reference.

2.4 Opravy jmen a adres autorů

Na opravu jmen a adres autorů jsem použil script:

```
git filter-branch --env-filter '
OLD_NAME="petr"
CORRECT_NAME="Petr"
CORRECT_EMAIL="petr@edgebox.net"
if [ "$GIT_COMMITTER_NAME" = "$OLD_NAME" ]
then
export GIT_COMMITTER_NAME="$CORRECT_NAME"
export GIT_COMMITTER_EMAIL="$CORRECT_EMAIL"
fi
if [ "$GIT_AUTHOR_NAME" = "$OLD_NAME" ]
then
export GIT_AUTHOR_NAME="$CORRECT_NAME"
export GIT_AUTHOR_EMAIL="$CORRECT_EMAIL"
fi
' --force --tag-name-filter cat -- --branches --tags --all
```

který jsem našel zde.

Jen jsem si musel dohledat pomocí `git log`, která jména jsou špatně.

2.5 Odevzdání fáze 2

Byly použity příkazy:

```
git checkout -b phase2
git push --set-upstream --force origin phase2
```

Fáze 3 Příprava na vydání

3.1 F*** bomba

Byl použit příkaz:

```
git filter-branch --tree-filter "git grep -l 'fuck' | xargs
sed -i '' -e 's/TODO.*fuck.*$/TODO/g'" -f --tag-name-filter cat -- --all HEAD
```

který jsem už pomocí předchozích příkazů a znalostí z PS1 vymyslel.

3.2 Smazání API klíčů

Byly použity příkazy:

```
cp js/keys.js js/keys.js.backup
git filter-branch --force --tree-filter '
git rm -f --ignore-unmatch js/keys.js' --tag-name-filter cat -- --all HEAD
mv js/keys.js.backup js/keys.js
```

které nejdříve skopírují soubor `js/keys.js`, následně smaže tento soubor že všech commitů a uloží soubor lokálně.

3.3 Oštítkování verze 0.1.0

Byl použit příkaz:

```
git log --patch -- VERSION
```

který vrátil adresu commitu, kde byl změněn soubor z verze `0.0.1_testing` na `0.1.0`.

```
git checkout 6cdf14c9850fa49a6a27fea90853e41a45a54da8
git tag v0.1.0
git checkout master
```

tag a checkout znám z přednášek.

3.4 Vývojářská větev

Byly použity příkazy:

```
git branch dev
git reset --hard v0.1.0
git checkout dev
git push --force origin dev
git push --force origin dev --tags
git push --force origin master
git push --force origin master --tags
```

Pro jistotu jsem udělal push ještě s přepínačem `-tags`.

Inspirace ze stackoverflow.

3.5 Vývojářská větev

Byly použity příkazy:

```
git branch dev
git reset --hard v0.1.0
git checkout dev
```

3.6 Nasazení hlavních větví repozitáře

Byly použity příkazy:

```
git push --force origin dev
git push --force origin dev --tags
git push --force origin master
git push --force origin master --tags
```

Pro jistotu jsem udělal push ještě s přepínačem `-tags`.

Inspirace ze stackoverflow.

Fáze 4 Vychytávky

4.1 Špatné konce řádků

Jeden ze způsobů, jak to vyřešit je příkaz `git config`.

Postup se liší podle toho, na kterém OS se nacházíme:

1. Na *Linux/OSX*, provedeme příkaz:

   ```
   git config --global core.autocrlf input
   ```

   Tento příkaz nám převede jakékoliv `CRLF` na `LF` při `commitu`.

2. Na *Windows*, provedeme příkaz:

   ```
   git config --global core.autocrlf true
   ```

   Tento příkaz nám při `checkoutu` převede `LF` na `CRLF` a při `commitu` zpátky `CRLF` na `LF`.

Tyto příkazy jsem zjistil ze stackoverflow.

4.2 Výpis TODO při commitu

Toto bych řešil pomocí `git hook`.

Nejdřív je za potřebí soubor, do kterého zadáme script, který se bude volat po každém commitu.

```
vim .git/hooks/post-commit
```

Poté do něj vložíme nějaký script. Nejspíše by stačilo mít nejaký soubor, ve kterém budou vývojáři při každém commitu připisovat nějaké TODO.

```
#!/bin/bash
vim TODO.txt
git commit -m "update TODO list"
```

Na závěr už jen stačí soubor post-commit udělat executable:

```
chmod +x .git/hooks/post-commit
```

Inspirace: zde.