Dokumentace k semestrální práci předmětu BI-GIT

Mykyta Boiko (boikomyk)

9. června 2018

Fáze 1 Slití repozitářů

1.1 Klon repozitáře

```
git clone fit@gitc.cz:LS2017/repo/boikomyk_sem.git
```

1.2 Klon SVN repozitáře

```
git svn clone https://exh.extremehost.cz/svn/edgebox/
```

1.3 Struktura repozitářů a postup slití

Byly použity příkazy:

- Pridame novy vzdaleny repozitar do repozitare s semetralni prace.
  Pojmenujeme novy repozitar svn_rep.
  Pomoci prikazu fetch ziskame data ze vzdaleneho repozitare, zatim nedelame merge.

  ```
  git remote add svn_rep ../edgebox/
  git fetch svn_rep
  ```

- Vypiseme graf-strukturu repozitare.

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

  Vysledek vypisu:

  * c0ca877 (HEAD → master, origin/master, origin/HEAD)
  * c1687f0 Prepare container for sorted images
  | * 50a7e0a (svn-project/master) FIX comments
  | * 6649f0a Actually sort images by date in descending order
  | * 9f69396 Display result container
  |/
  6493a7a Correctly offset the author container
  atd...

- Zkontrolujeme pocet commitu. Celkem ma byt 31 commit.

```
git shortlog -s -n
nebo
git rev-list --all --count
```

Vysledek vypisu:

31

1.4 Rebase

Byl použit příkaz:

- Prepneme se na master a provedeme dalsi prikaz:

  ```
  git rebase svn_rep/master
  ```

- Tim vezmeme vsechny zmeny, ktere byly zapsany na master vetvi, a nechame je znovu provest na svn_rep/master vetvi. Strucne receno nakopirujeme vsechny commity z master vetvi do svn_rep/master vetvi.

- Pri rebasu dostavame 2 konfilkty. Rucne to resit nebudeme, k tomu muzeme vyuzit kterykoliv nastroj pro merge. V nasem pripade je to Meld.

  ```
  git mergetool
  git rebase --continue
  ```

- Vypiseme jak vypada novy graf:

  ```
  * e97e030 (HEAD -> master) Commit gitignore rep by another user
  * 0aa4857 Actually sort the images
  * 50a7e0a FIX comments
  * 6649f0a Actually sort images by date in descending order
  * 9f69396 Display result container
  | * c0ca877 (origin/master, origin/HEAD) Actually sort the images
  | * c1687f0 Prepare container for sorted images
  |/
  * 6493a7a Correctly offset the author container
  ```

1.5 Smazání reference na SVN

Byl použit příkaz:

- Smazeme referenci na vzdaleny svn repozitar.

  ```
  git remote rm svn_rep
  ```

- Zkontrolujeme zmeny pouzitim dalsich prikazu:

  ```
  git remote show
  git remote --verbose
  ```

  Vysledek:

```
(first command):
origin
(second command):
origin fit@gitc.cz:LS2017/repo/boikomyk_sem.git (fetch)
origin fit@gitc.cz:LS2017/repo/boikomyk_sem.git (push)
```

1.6 Návrh .gitignore

Byl vytvořen nový soubor `.gitignore`:

- V kteremkoliv editru vytavirm soubor .gitignore.
- Zadame "*∼"a ulozime zmeny do souboru.
- Zadany radek v souboru specifikuje pattern. Pri rozhodovani, zda ma cesta ignorovat, Git kontroluje obsah gitignore patterny. Docasne soubory, ktere obsahuji na konce symbol "∼"budou ingorovany. To ale neplati pro jiz existujici soubory.

1.7 Commit pod jiným uživatelem

Byly použity příkazy:

```
git add .gitignore
git commit --author "Petr < petr@edgebox.net >" -m "Commit gitignore rep by another user"
```

1.8 Pročištění historie od ignorovaných souborů

Byly použity příkazy:

```
git filter-branch --force --tree-filter 'find . -name "*~" -type f -exec rm -rf {} \;'
-- --all
```

1.9 Pročištění referencí

- U vsech dalsich kanonu byl pouzit prepinac –force.

1.10 Odevzdání fáze 1

- Vytvorime novou vetev phase1 s nasledujicim push na server.

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

Fáze 2 Čištění repozitáře

2.1 FIX of FIX

Byly použity příkazy:

- Prepneme se na master vetev:

  ```
  git checkout master
  ```

- Provedeme nasledujici prikaz:

  ```
  git rebase -i HEAD~31
  ```

- Interaktivni rebase dolovuje nam zvolit jak commity budou rebased.
- V zobrazenem souboru s commity prepiseme pick na squash u commitu s nazevem "Fix of Fix".

```
pick 8087ac0 Basic user interface
pick b4f37a1 Create prototype of EdgeBox
pick 8ae3296 0.0.1 Release
pick d5564db FIX comments and typos
squash a536325 FIX Commenting CSS
squash 0309498 FIX Comments
squash d3affe6 FIX Yet another comments
pick cb0d478 Enable selecting how many images are returned
pick 4eebc9a Enable user to chose size of thumbnails
pick e16506e Propose UI for sorting by distance
pick 8cfc90e FIX Add js for map initialization
squash b525164 FIX add style for UI
squash 2ed9b07 FIX of FIX, add images also
pick e22a832 Enable click to select coordinates
pick 8c50be9 Add geocoding (zoom to address)
pick 3d939d7 Show "By GPS" section only if some point selected
pick ca56acc Count distance and store in image element
pick 7525981 Enable user to force presence of geo information in all pictures
pick f7db6f5 Actually sort the container
pick 1dcd381 FIX Translated messages
squash c345d30 FIX of FIX in documentation
pick 14e95fd Better documentation
pick 7446dfc Enable sorting by author
pick 0b7ca67 User interface for sorting by date and size
pick 6493a7a Correctly offset the author container
pick 9f69396 Display result container
pick 6649f0a Actually sort images by date in descending order
pick 50a7e0a FIX comments
pick 67f9c0b Prepare container for sorted images
pick fd26c02 Actually sort the images
pick a6d5335 Commit gitignore rep by anoth
```

2.2 Windows konce řádků

Byly použity příkazy:

- Pouzjieme k tomuinterpretovaný jazyk perl. Prikaz bude mit nasledujici syntaxi:

  `perl -pi -e 's/pattern/ReplaceWith/g' inputFile`

- Celkem vyraz vapada tahle:

```
git filter-branch -f --tree-filter "test \"index.html\"
&& perl -pi -e 's/\r\n/\n/g' index.html " -- --all
```

- Pomoci uvedeneho prikazu smazeme symbol "\r"ze souboru index.html.

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

Byly použity příkazy:

- K tomu vyuzijeme prepinac –msg-filter, ktery slouzi pro prepsani zprav commitu.
- Jeste budeme potrebovat dalsi prepinac "–tag-name-filter cat"pro update stiku a prave prepojeni stiku na nove vznikle commity.
- Cely prikaz bude vypadat nasledovne:

```
git filter-branch -f --msg-filter 'perl -i -pe "s/git-svn.*$//g"'
--tag-name-filter cat -- --all
```

2.4 Opravy jmen a adres autorů

Byly použity příkazy:

- K vyreseni dalsiho problemu budeme potrebovat napsat maly bash script, s volanim promennych prostredi git, takovy jak: GIT_AUTHOR_EMAIL, GIT_AUTHOR_NAME.
- Vytvorime bash script s nasledujicim obsahem a spustime:

```
git filter-branch -f --commit-filter '

#Nastavime promenne pro ziskani domainu a jmena z mailu:

CORRECT_DOMAIN="edgebox.net";

ACTUAL_DOMAIN=$(echo "$GIT_AUTHOR_EMAIL" | awk -F  "@"  "{print $2}");
NAME=$(echo "$GIT_AUTHOR_EMAIL" | awk -F  "@"  "{print $1}");

GET_AUTHOR_NAME=$(echo "$GIT_AUTHOR_NAME");
SET_AUTHOR_NAME=$(echo "$GIT_AUTHOR_NAME");
SET_AUTHOR_NAME="$(tr '[:lower:]' '[:upper:]'
<<< ${SET_AUTHOR_NAME:0:1})${SET_AUTHOR_NAME:1}"

#Provedeme vsechny korekce:

if [ "$ACTUAL_DOMAIN" != "$CORRECT_DOMAIN" ]
then
GIT_AUTHOR_EMAIL="$NAME@$CORRECT_DOMAIN";
fi

if [ "$GET_AUTHOR_NAME" != "$SET_AUTHOR_NAME" ]
then
GIT_AUTHOR_NAME="$SET_AUTHOR_NAME";
```

```
25  fi
26
27  git commit-tree "$@";
28  ' --tag-name-filter cat -- --all
```

Zdroj nadseni:
GitHubGist Question

2.5 Odevzdání fáze 2

- Vytvorime novou vetev phase2 a udelame push.

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

Fáze 3 Příprava na vydání

3.1 F*** bomba

Byly použity příkazy:

- Prepneme se na master vetev.

  ```
  git checkout master
  ```

- Napiseme bash script s nasledujicim obsahem:

```
1  git filter-branch -f --tree-filter '
2  files="$(grep -RHl "fuck" *)";
3  for file in $files;
4  do perl -pi -e "s/\/\/.*fuck.*/\/\/TODO/"  $file;
5  done' --tag-name-filter cat -- --all
```

- Spustime bash script.

3.2 Smazání API klíčů

Byly použity příkazy:

- Udelame si lokalni kopii souboru keys.js.

```
1  cp js/keys.js js/cp_keys.js
```

- Smazeme tento soubor ze vsech revizi.

```
1  git filter-branch -f --tree-filter '
2  rm -f js/keys.js' --tag-name-filter cat -- --all
```

3.3 Oštítkování verze 0.1.0

Byly použity příkazy:

- Najdeme revizi, ve ktere doslo ke zmene verzi.

  ```
  git log -p VERSION
  ```

- Dostavame vypis

  ```
  6b3f180
  diff --git a/VERSION b/VERSION
  index bd2e853..6e8bf73 100644
  --- a/VERSION
  +++ b/VERSION
  @@ -1 +1 @@
  -0.0.1_testing
  +0.1.0
  ...
  ```

- Doslo ke zmene verzi v commitu s hashi 6b3f180.

- Ostitkujeme prislusny commit:

  ```
  git tag -f "v0.1.0" 6b3f180
  ```

3.4 Vývojářská větev

Byly použity příkazy:

- Potrebujeme vytvorit novou vetev dev od aktualni master vetvi a udelat rollback do revizi specifikovanou hashi nastavenem par kroku predem.

  ```
  git branch dev
  #Rollback
  git checkout v0.1.0
  ```

- Dale smazeme vetev master a vytvorime novou
  (z commity se stitkem "v0.1.0").

  ```
  git branch -D master
  git branch master
  ```

3.5 Nasazení hlavních větví repozitáře

Byly použity příkazy:

- Nahrajeme na server obedve vetvi: master a dev.

- git push prikaz neprenasi stitky na server. Proto budeme potrebovat pouzit prepinac –tags, ktery umozni prenest vsecnhy stitky na server, ktere tam nejsou.

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

Fáze 4 Vychytávky

4.1 Prevence špatných konců řádků

- Windows pouziva obadva: carriage-return a linefeed symboly pro odradkovani ve svych souborech. Mac and Linux systemy pouzivaji pouze linefeed symbol.
- Spousta editoru na Windows nahrazuje existujici LF-style konce radku CRLF-stylem.
- Git ma nekolik moznosti konfigurace, ktere nam pomohou s temito problemy.
- V repozitare bude konec radku vzdy reprezentovat v LF stylu.

  core.autocrlf

  1. Pri operacnim systemu Windows:

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

     true – konvertuje LF do CRLF pri stazeni. Pak pri nahrani zmeni zpatky z CRLF do LF.

  2. Pri operacnim systemu Linux/MAC:

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

     Takove nastaveni musi nam nechat CRLF konce radku ve Windows, LF konce radku na Mac a Linux systemech a i dokonce v repozitare. Nahodou vznikly soubor s CRLF koncem radku bude convertovan do LF pri commitu.

  3. Operacni system Windows s Windwos-only projektem:

     `$ git config --global core.autocrlf false`

     false - vypnout funkcionalitu. Zapisovat carriage returns do repozitare.

Zdroj nadseni:
https://github.com/Microsoft/WSL/issues/2318

4.2 Výpis TODO při commitu

- Potrebujeme přizpůsobit vnitřní chování Git. Proto vsechny zmeny vneseme do Git Hooks, kde vsechny scripty budou spusteny automaticke pri vzniku udalisti v Git repoziraru. A prave do post-commit, kde script bude spusten po commitu.
- Z predchozich ukolu, vime, ze komentar "TODO"se muze vyskytovat v nekterych souborech. Nas ucel je projit vsechny soubory po commitu a najit aktualni TODO komentare.
- Vytvorime novy script a ulozime jej do .git/hooks/post-commit.
- Script bude mit nasledujici obsah:

```
1  #!/bin/bash
2  echo "TODO LIST:";
3  todoList="$(grep --exclude=\*.bash -RHl 'TODO' *)";
4  TASKS_COUNT=$(wc -l <<< "${todoList}")
5  echo "Contains number of tasks: " $TASKS_COUNT;
6  index=1;
```

```
7  for FILE in "$todoList";
8  do
9    comment="$(grep -n 'TODO' $FILE)";
10   echo -e "$index) File: $FILE";
11   line=$(printf "$comment" | cut -d':' -f1);
12   body=$(printf "$comment" | cut -d':' -f2 | cut -d'*' -f2 | sed -E 's/^ +//g');
13   echo -e "\tLine: $line";
14   echo -e "\tBody: $body";
15   index="$index"+1;
16 done
```

Priklad vypisu (Po kazdem commitu se bude vypisovat):

```
TODO LIST:
Contains number of tasks:  1
1) File: js/edgebox.js
Line: 175
Body: TODO - delta calculation implemented is not optimal yet faster viz.
...
```