Technical University of Moldova

CIM Faculty

FAF

Report

Integrated Development Environments

Laboratory work #1

Done by: student, gr. FAF-151

Bîzdîga Stanislav

Verified by: Patrasco Alex

Chisinau 2017

Topic: Setting server environment. Version Control Systems

Prerequisites:

- IDEs: Command Line Interface (CLI) | CLI Editors - nano, vim, emacs
- Languages: bash, C/C++
- Technologies: Version Control Systems | The power of vim

Objectives:

- Understanding and using CLI (basic level)
- Administration of a remote linux machine using SSH (remote code editing)
- Ability to work with Version Control Systems (git || mercurial || svn)
- Compile your C/C++/Java/Python programs through CLI using gcc/g++/javac/python compilers

Tasks:

- Mandatory tasks(mark = 5):
  - connect to a server using SSH
  - run at least 2 sample programs from provided HelloWorldPrograms set
  - configure your VCS
  - initialize an empty repository
  - create branches (create at least 2 branches)
  - commit to different branches (at least 1 commit per branch)
- Optional tasks:
  - Intermediate(each task is worth .5 points):
    - set a branch to track a remote origin on which you are able to push
    - reset a branch to previous commit
    - merge 2 branches
    - resolve a conflict
    - create a meaningful pull request
  - Advanced(each task is worth 1 point):
    - GIT cherry-pick
    - GIT rebase
    - GIT hooks
    - Write a script that will compile any chosen project from the list of HelloWorldPrograms projects. Make your script output compilation results for each project
    - For each failed attempt do one of the following:
      - send an e-mail (to preset address) with compilation details
      - post a ticket to an issue tracker. Add compilation details

Work processing

Mandatory tasks:

1) Connect to a server using SSH:

To connect to the github server, we need to generate an SSH key and then attach it to the ssh-agent. In order to do that, I used git-bash terminal.

I wrote the following command:

```
$ ssh-keygen -t rsa -b 4096 -C "stanislav.bizdiga@faf.utm.md"
```

It generated a new ssh key using the provided email.
Then I was prompted to "Enter a file in which to save the key," I pressed Enter to accept the default file location. Then this followed:

```
$ Enter passphrase (empty for no passphrase): [Type a passphrase]
$ Enter same passphrase again: [Type passphrase again]
```

Here I wrote a secure password to keep my account access private.

Next, because I use git-bash, ssh-agent is running by default, but if we need to start it anyway, just type the following command:

```
$ eval $(ssh-agent -s)
```

It will return something like: `Agent pid 59566` , proving it's all up now.

Lastly, I add the ssh key to my github account. This is done by copying my ssh key, using command:

```
$ clip < ~/.ssh/a.pub
```

Note: a.pub is the name of the file I saved my ssh key into.
Alternatively I could simply open the file with a text editor and copy its content.

Then I went to github.com -> settings -> ssh and gpg keys -> add ssh key
There in the Key field, I pasted my ssh key I copied. I added the ssh key by pressing the button.

Now I can access my account from terminal without having to log in github every time.
I just need to remember my pass-phrase. :)

2) Configure your VCS

Configuring the VCS is pretty simple. All I need to do is write in git-bash my git user-name and email:

```
$ git config --global user.name "StasBizdiga"
$ git config --global user.email stanislav.bizdiga@faf.utm.md
```

Optionally I could change the default text editor by the command:

```
$ git config --global core.editor emacs
```

This command sets the default text editor to emacs, but I prefer vim, so I skipped this option.
For more configurations I can type:

```
$ man git-config
```

This would show me what other commands git-bash recognizes.

3) Run at least 2 sample programs from provided HelloWorldPrograms set

It was a tough one. I needed first to get my compilers, since I am on Windows. So I downloaded the required compilers and added them to the PATH environment variable so that terminals could recognize the short commands like gcc, g++, javac, python, etc.

First, I cloned the IDE repository by typing: `git clone https://github.com/TUM-FAF/IDE`
That granted me access to the hello programs I need. Then I tried the following in git bash:
I accessed the folder HelloWorldPrograms using the cd <filepath> command. Then, I do:

```
cd c
vim hello.c
```

And vim opens the file:

```c
#include <stdio.h>

int main()
{
  printf("Hello world\n");
  return 0;
}
```

And now I want to compile it so I do:
<escape> (to go to *normal mode* in vim, because it may start in *insert mode*)
:! clear; gcc -o hello % && ./hello

And now I get:

```
Hello world

Press ENTER or type command to continue
```

By pressing enter I go back to vim. Then I close the file with :q, or :wq to save and quit.

Similarly, with cpp:
In bash:

```
cd ..
cd cpp
vim hello.cpp
```

Then I see it in vim:

```cpp
#include <iostream>

using namespace std;

int main()
{
  cout << "Hello World!" << endl;
  return 0;
}
```

In vim:
<escape>
:! g++ -o hello % && ./hello
And I got:

```
Hello World!

Press ENTER or type command to continue
```

Let's try python too:
Bash:

```
cd ..
cd python
vim hello.py
```

```
print "Hello World!"
```

Vim:
:! python %

```
Hello World!

Press ENTER or type command to continue
```

4) Initialize an empty repository

As simple as:

```
mkdir MIDPS
cd MIDPS
git init
git add *
git commit -m "My first commit"
git remote add origin git@github.com:StasBizdiga/MIDPS.git
git push -u origin master
```

5) Create branches (create at least 2 branches)

Main commands:

```
git branch - lists all branches
git branch <name> - creates a branch with given name
git branch -m <name> - renames current branch to name
git branch -d <name> - safely deletes the given branch (stops if unmerged changes)
git branch -D <name> - force delete given branch (deletes even if unmerged changes)
```

Creating two branches:

```
git branch first
git branch second
```

6) Commit to different branches (at least 1 commit per branch)

```
git checkout <name> - command to go to the given branch
```

```
git checkout first
vim README.txt    (writing changes in there)
git add README.txt
git commit -m "commit on branch 1"
```

```
git checkout second
vim README.txt    (changing README.txt text data)
git add README.txt
git commit -m "commit on branch 2"

git checkout master
```

Optional tasks

Intermediate:

1) Set a branch to track a remote origin on which you are able to push

```
$ git push -u origin first
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 300 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
Branch first set up to track remote branch first from origin.
To github.com:StasBizdiga/MIDPS.git
   43eeaae..4fb7ce3  first -> first
```

2) Reset a branch to previous commit

$ git checkout second

```
nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (second)
$ git log
commit 799913e8688152ee95547d95d51e27a21172cf4f
Author: StasBizdiga <azurium.official@gmail.com>
Date:   Tue Mar 14 23:38:48 2017 +0200

    commit on branch 2

commit 43eeaae5e6eb94c879d47e47f65f57c4e59d71ed
Author: StasBizdiga <azurium.official@gmail.com>
Date:   Wed Feb 15 09:39:13 2017 +0200

    Lab_1

commit 67834d1c3aff74d2fffab326443af8519cf532fe
Author: StasBizdiga <azurium.official@gmail.com>
Date:   Wed Feb 15 09:29:32 2017 +0200

    first

nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (second)
$ git reset --hard HEAD
HEAD is now at 799913e commit on branch 2
```

3) Merge 2 branches

```
nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (first)
$ git checkout second
Your branch is ahead of 'origin/second' by 1 commit.
  (use "git push" to publish your local commits)
Switched to branch 'second'

nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (second)
$ git merge first
Updating 799913e..6c105ee
Fast-forward
 README.txt | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

4) Resolve a conflict

```
nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (first)
$ git merge second
Auto-merging README.txt
CONFLICT (content): Merge conflict in README.txt
Automatic merge failed; fix conflicts and then commit the result.

nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (first|MERGING)
$ vim README.txt
```

README.txt:

```
<<<<<<< HEAD
First was here!
=======
Oh it's empty! Parallel universes rule.

>>>>>>> second
~
```

README.txt | After editing:

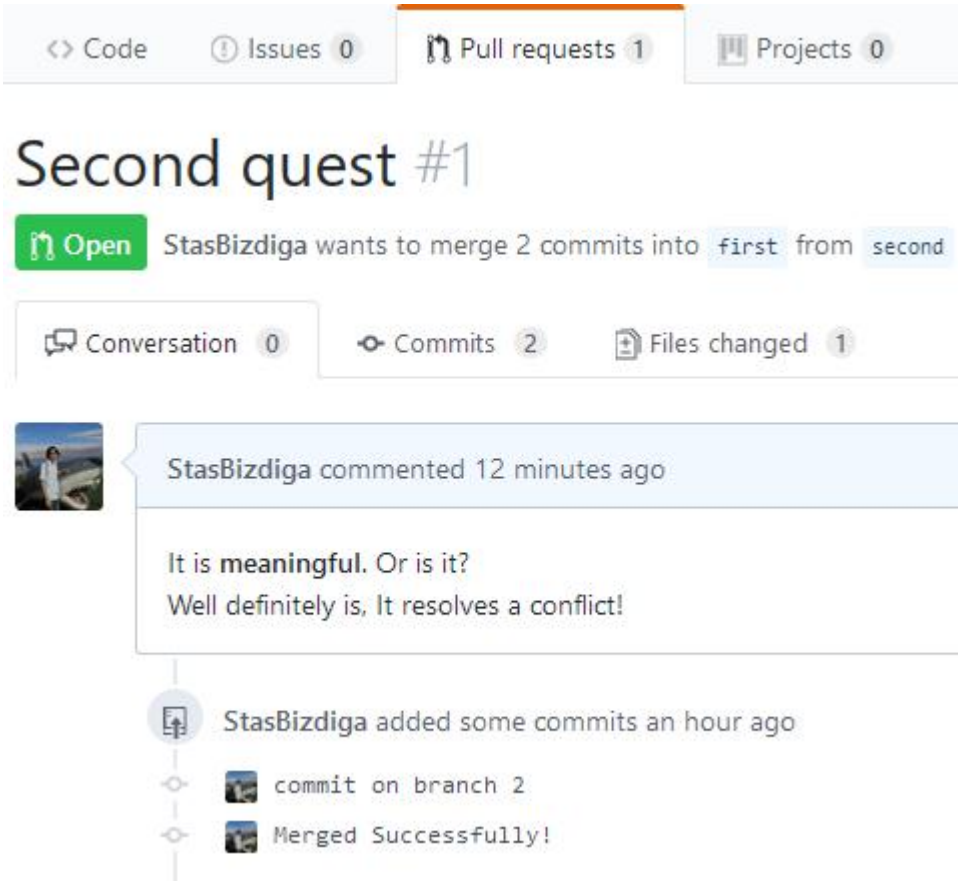
```
Parallel universes rule.
~
~
```

Once resolved the conflict:

```
nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (first|MERGING)
$ git add README.txt

nato2@LAPTOP-8S0HTRO6 MINGW64 /c/Github/MIDPS (first|MERGING)
$ git commit -m "Merged Successfully!"
[first 6c105ee] Merged Successfully!
```

5) Create a meaningful pull request



Pull requests are used for examining the changes between branches and ultimately, merging them to the master branch. It is easily possible to be commented on every line of code or just studied around in the files and lines that have been changed in the branch, along with also the specific commits performed.

Conclusion

In this laboratory work, I've learned a lot about Version Control Systems, specifically - GitHub. It provides a great opportunity to manage your projects.
Also, I learned working with Command Line Interfaces and editors. Vim is a great tool combined with bash. The speed of operating within a file is significantly higher once getting used to the commands and keys that vim is based on.
And finally, common compilers are interestingly integrated by default in OS like Unix, Ubuntu, Linux and others, while not in Windows, you need to have headache with separate installations of MinGW, python 2.7, Java etc. - which is not very intuitive.

My repository: https://github.com/StasBizdiga/MIDPS