Mandatory tasks:

- connect to a server using SSH

1. Install a ssh server on a machine. In our case we will install openssh server on a unix machine using command.

```
sudo apt-get install openssh-server
```

2. Now we need to configure the server so first time we will be able to connect only by using password. To do so we will edit the configuration file.

```
vim /etc/ssh/sshd_config
```

Here we will set the option

#PasswordAuthentication yes

to

PasswordAuthentication yes

In this way we will be able to connect to the server using only the user password.

3. After this we will restart the server so that the configuration file will take effect by using.

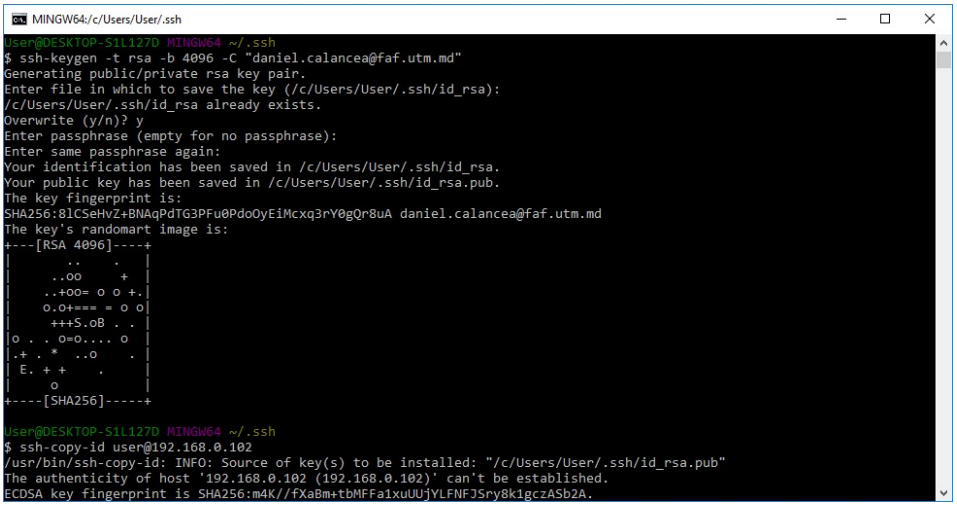
```
sudo restart ssh
```

or in case of error on this command

```
sudo systemctl restart ssh
```

4. Before we connect to our server we will generate on the device we want to connect from the RSA key pair using command.

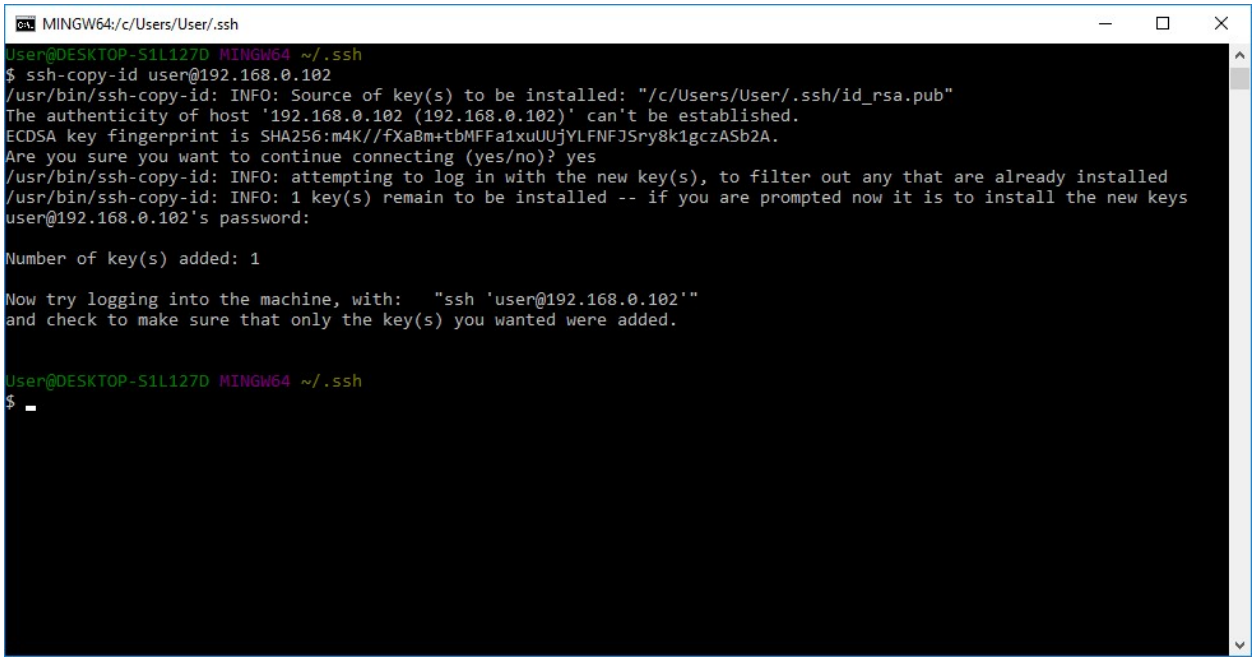
```
ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
```



5. Now we need to add the ssh public key to our server so that we will be able to connect next time without using any passwords. We will use:

```
ssh-copy-id <username>@<host>
```

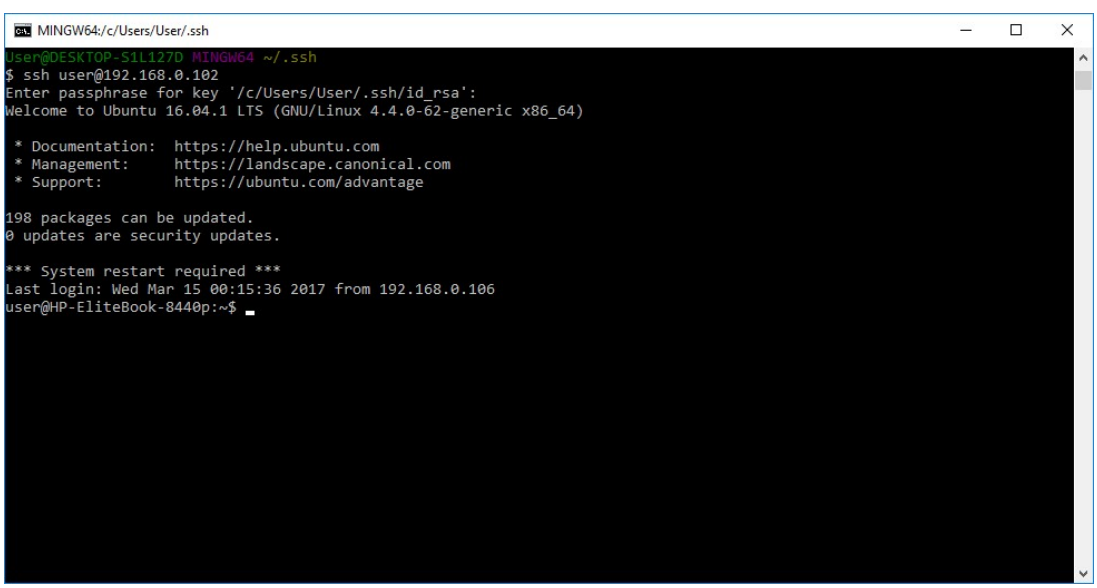
In our case it will be `ssh-copy-id` `user@192.168.0.102` .This is local ip because both of our devices are connected to the same router.



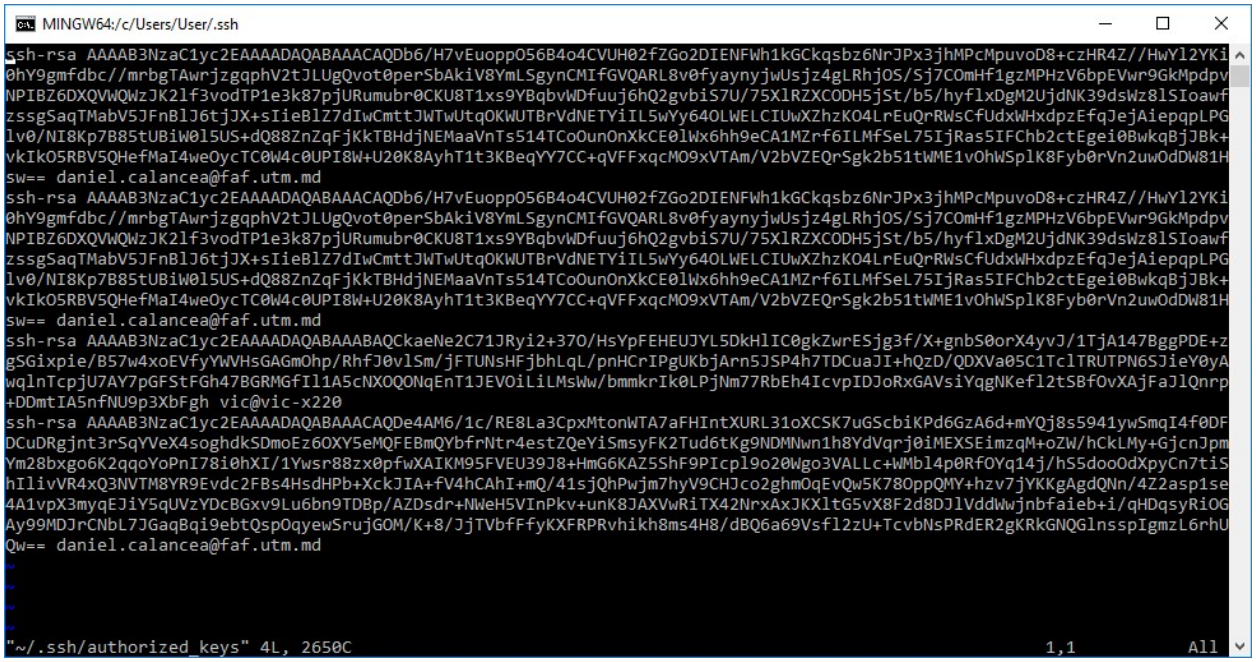
We connected by using the username and password that were used by Ubuntu.

6. Now we can check if our key was added. We can do this by connecting to the machine and viewing file ~/.ssh/authorized_keys. We will use command

```
ssh user@192.168.0.102
```



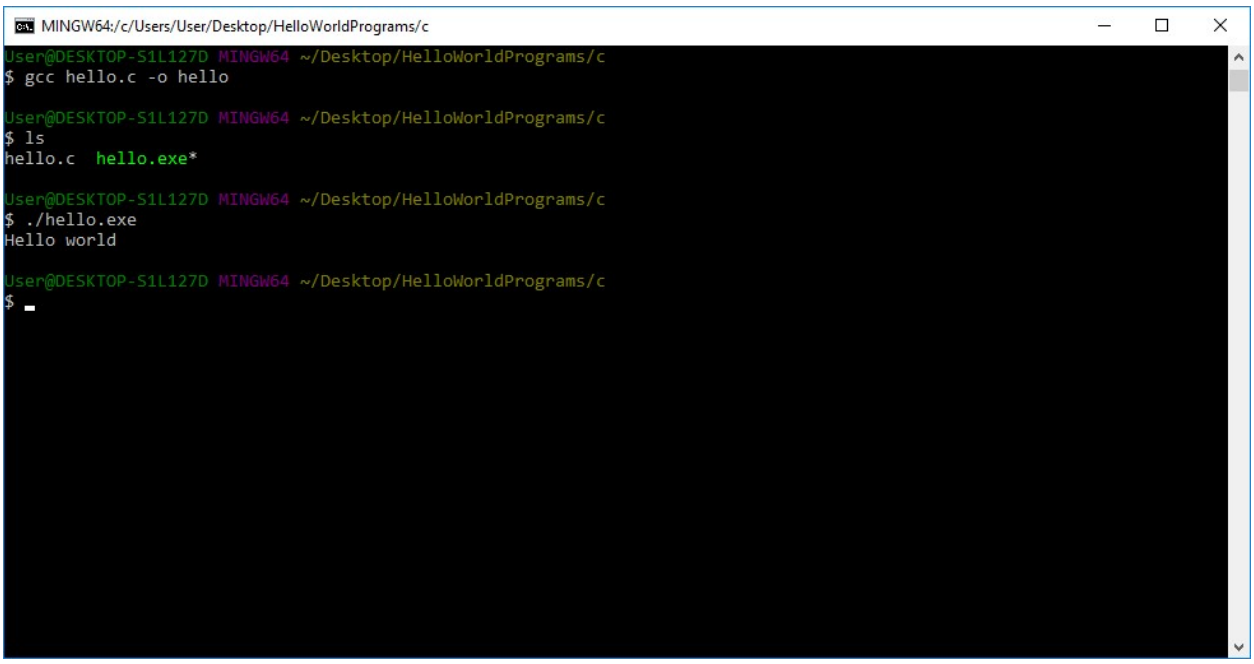
7. We can see that it connected using our ssh key, even though we didn't disable the connection using password in sshd_config yet. If you are not able to connect at this moment, then you need to check and make sure that:
   #AuthorizedKeysFile %h/.ssh/authorized_keys
   Is
   AuthorizedKeysFile %h/.ssh/authorized_keys
   In sshd_config file



8. We can observe that the last key is the one corresponding to our device.
9. Now we can disable the password authentication and only the users with public keys listed in this file will be able to connect to our server.

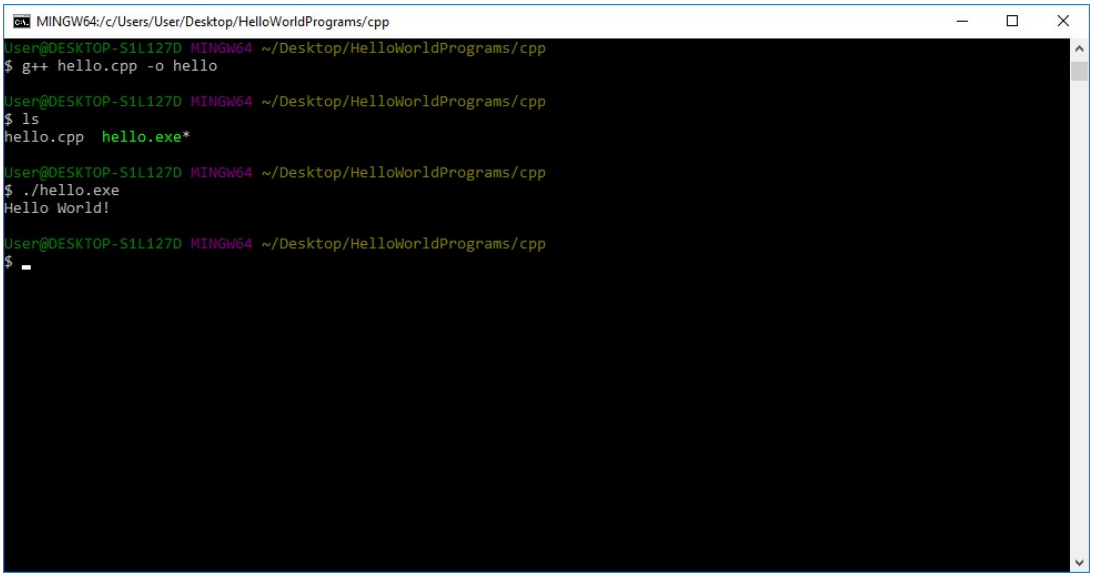
- run at least 2 sample programs from provided HelloWorldPrograms set

1. We compile the C program first. I am using git bash on windows that is why this process is a little different. I installed gcc packages for bash to be able to compile this program and it generates a exe file, that is different from the file it generates on linux.

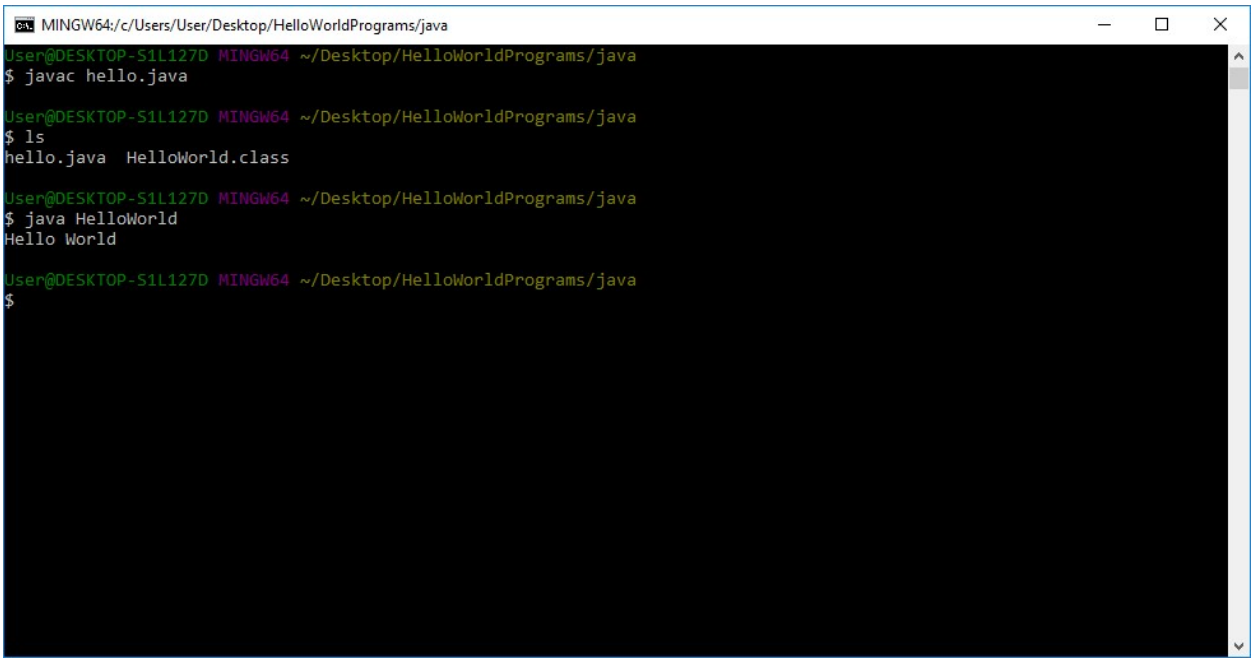


Here we can observe that it compiled and executed with success.

2. Compiling C++ program.



3. Compiling Java program. To be able to do this in windows we need to have jdk and also add the jdk path to system variable PATH.



- configure your VCS

1. First of all we need to know that we can make configuration for a specific repository and global configurations, in our case we will use git.
2. We will start by configuring global username and user email. By using

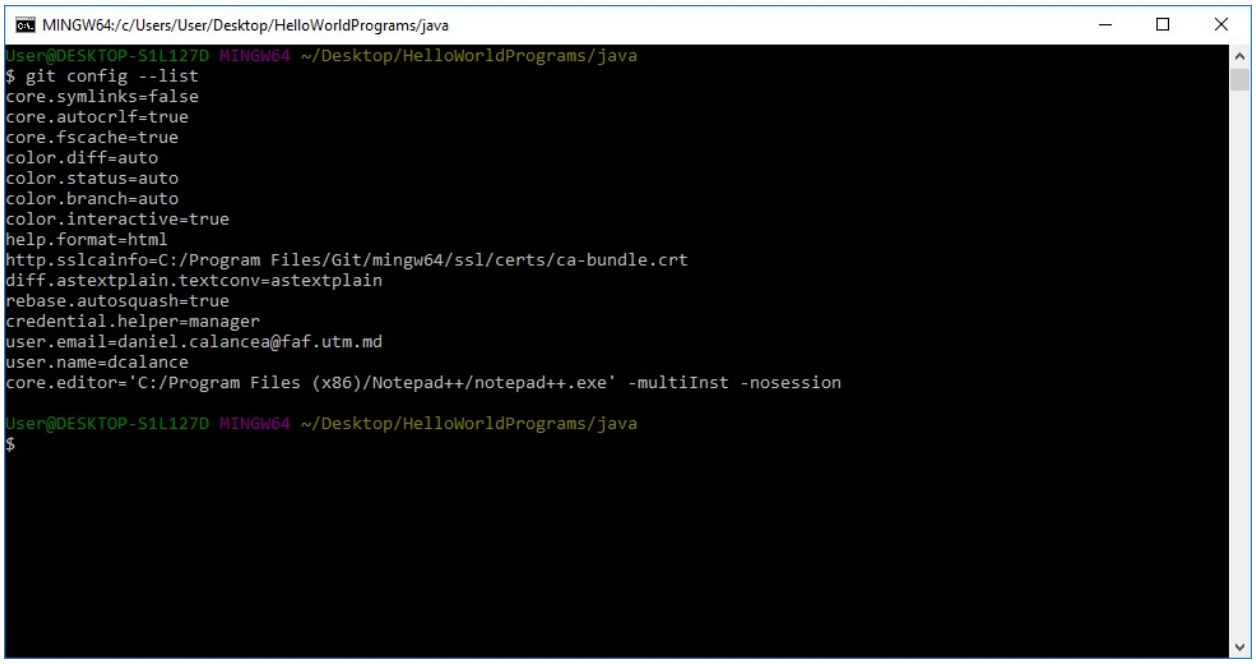
```
git config --global user.name "dcalance"
git config --global user.email daniel.calancea@faf.utm.md
```

3. Now we will configure our default text editor. I am using windows so I will use Notepad++ as my default text editor.

```
git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe'
-multiInst -nosession"
```

4. We can list all of our settings to see if they changed using:

```
git config --list
```

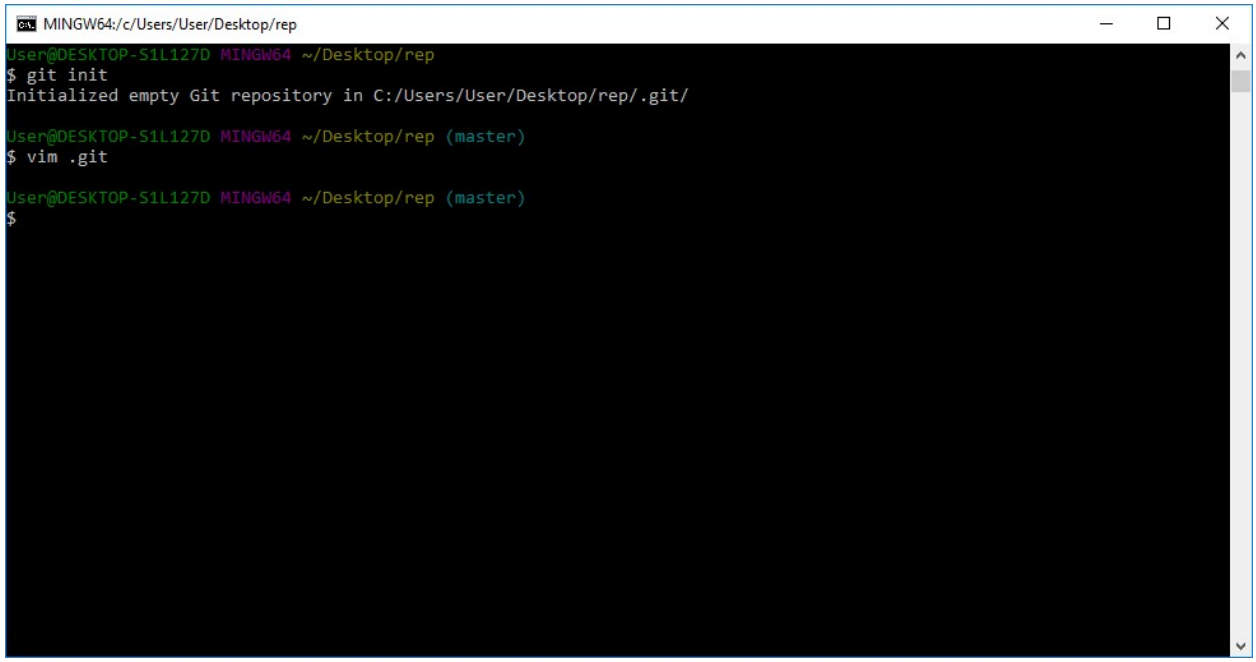


- initialize an empty repository

1. We select a folder where we want to create a repository and use the command: (make sure you have git installed)

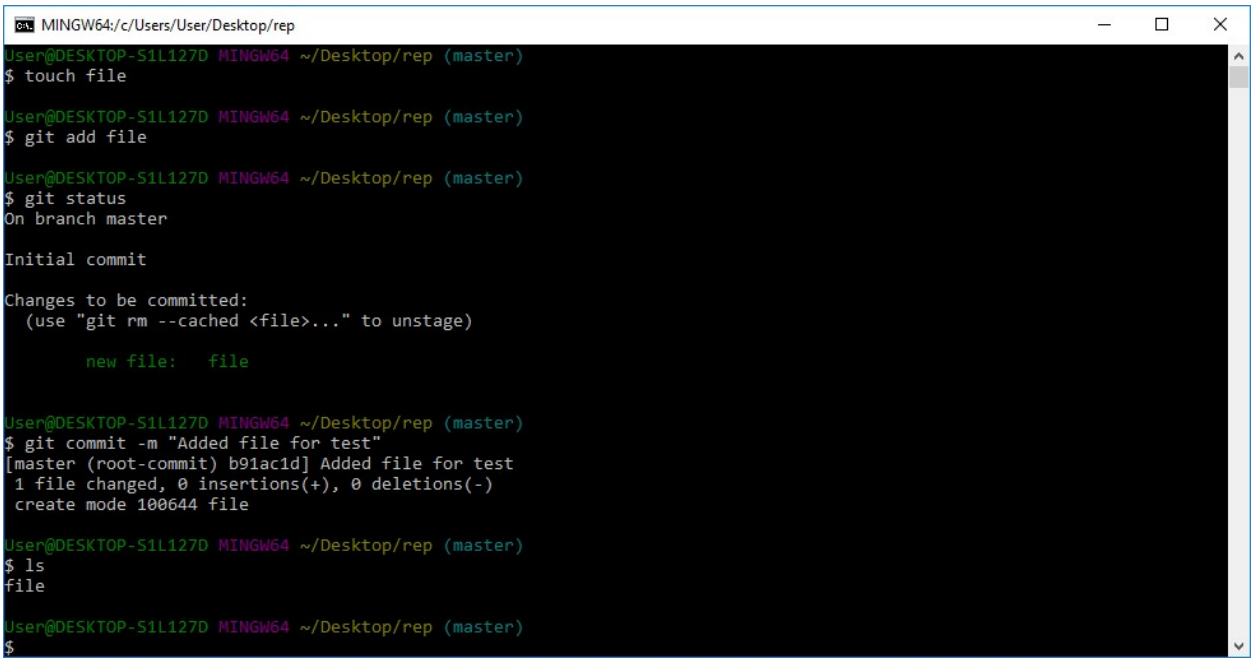
```
git init
```

2. We should get a message like **Initialized empty Git repository in yourpath/.git/ .** We can also check for .git file that should be in root folder of repository.



- create branches (create at least 2 branches)

1. First of all, we need to make our first commit in order to be able to track all of our branches. We will begin by creating a file and committing it on master branch.



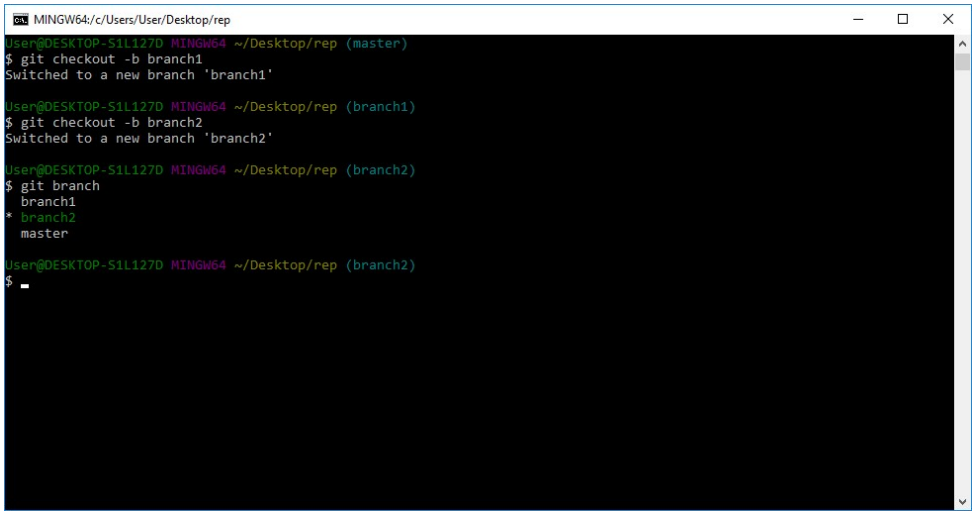
2. Now we can create a new branch using

```
git checkout –b branch1
```

It will automatically create a new branch and switch to it, also all content from master will be copied to the branch.

3. Using same method we create branch2 and list all of them using:

```
git branch
```



- commit to different branches (at least 1 commit per branch)

1. First make sure you are on the correct branch. In git bash I can see directly on what branch I'm located, but if you don't have this option you can always use git branch to see all branches and the one you are located will be highlighted.
2. Now we will make 2 commits on branch1.

```
MINGW64:/c/Users/User/Desktop/rep
User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ vim file2

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ vim file1

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git add file1
warning: LF will be replaced by CRLF in file1.
The file will have its original line endings in your working directory.

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git commit -m "Added file1"
[branch1 4561fce] Added file1
 1 file changed, 1 insertion(+)
 create mode 100644 file1

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git add file2
warning: LF will be replaced by CRLF in file2.
The file will have its original line endings in your working directory.

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git commit -m "Added file2"
[branch1 7998d64] Added file2
 1 file changed, 1 insertion(+)
 create mode 100644 file2

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ _
```

3. We will switch now to branch2 and modify file **file** and add a file3.

```
MINGW64:/c/Users/User/Desktop/rep
User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git checkout branch2
Switched to branch 'branch2'

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ vim file

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ ls
file

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ vim file3

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ git add file
warning: LF will be replaced by CRLF in file.
The file will have its original line endings in your working directory.

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ git commit -m "Modifications to file"
[branch2 3c65879] Modifications to file
 1 file changed, 1 insertion(+)

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ git add file3
warning: LF will be replaced by CRLF in file3.
The file will have its original line endings in your working directory.

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch2)
$ git commit -m "Added file3"
[branch2 36f3409] Added file3
 1 file changed, 1 insertion(+)
 create mode 100644 file3
```

Optional tasks:

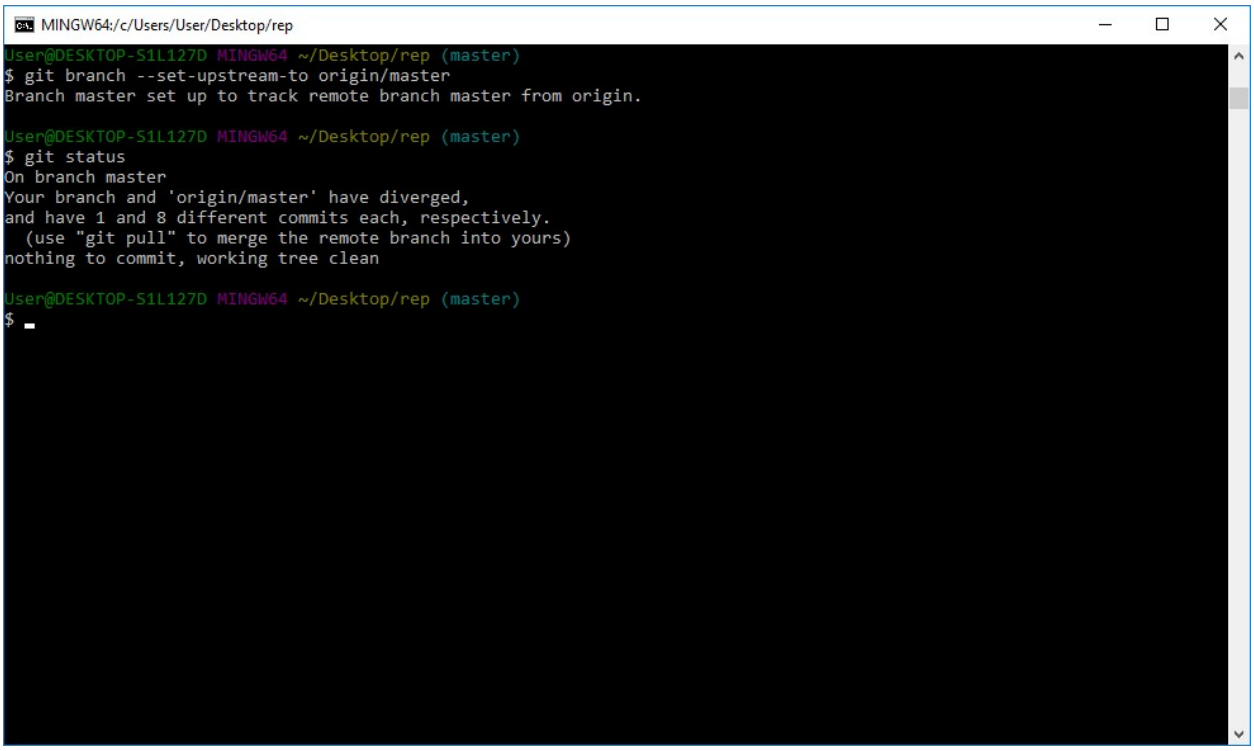
- set a branch to track a remote origin on which you are able to push (ex. Github, Bitbucket or custom server) and resolve a conflict.

1. First of all, we need to add our link to repository so that we will know how to locate repository.

```
git remote add origin git@github.com:dcalance/IDE.git
```

2. Now we need to set that our remote branch will track our master branch.

```
git branch --set-upstream-to origin/master
```



3. We can see that we already have conflicts since we pulled a file from a different repository and after that we changed the origin and tried to push a file with a different history. We can solve this by rebasing branches.

```
git rebase origin/master
```

4. Now we can push easily our content to our remote repository. We push only content on master branch because that's how we set it to.

```
MINGW64:/c/Users/User/Desktop/rep
User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (master)
$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 8 different commits each, respectively.
  (use "git pull" to merge the remote branch into yours)
nothing to commit, working tree clean

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (master)
$ git merge origin/master
fatal: refusing to merge unrelated histories

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (master)
$ git rebase origin/master
First, rewinding head to replay your work on top of it...
Applying: Added file for test

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (master)
$ git push
Enter passphrase for key '/c/Users/User/.ssh/id_rsa':
Counting objects: 54, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (44/44), done.
Writing objects: 100% (54/54), 39.02 KiB | 0 bytes/s, done.
Total 54 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), done.
To github.com:dcalance/IDE.git
 * [new branch]      master -> master

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (master)
$
```

- reset a branch to previous commit

1. We will go to our branch1 we created earlier and we will reset one commit to it. We can do this in 2 ways. We can use

```
git reset --hard “our commit id”
```

However we will lose track of the fact that we reversed a commit so we will use another option

```
git revert 7998d64
```

In this case the number is starting id of the commit. And as result we will get a new commit that will undo the previous one without deleting any commits, meaning that we can revert this revert in a way if we need that commit back. We can use view our commits using:

```
git log
```

MINGW64:/c/Users/User/Desktop/rep

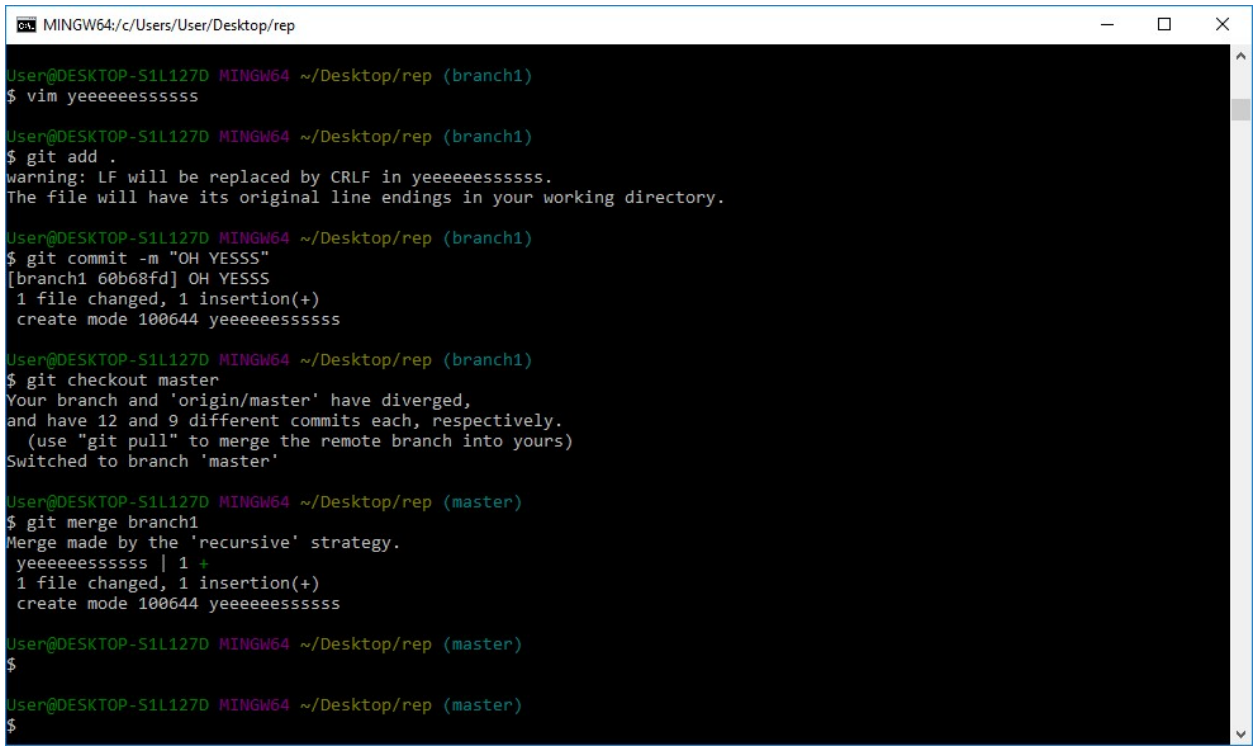
```
User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git revert 7998d64
[branch1 fc8abe0] Revert "Added file2"
 1 file changed, 1 deletion(-)
 delete mode 100644 file2

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git log
commit fc8abe0ec99e1ae450ebc3d989cf9e6699472626
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 02:32:45 2017 +0200

    Revert "Added file2"

    This reverts commit 7998d64d72345ea7ee6c336ee6b2732e8d741daa.

commit 7998d64d72345ea7ee6c336ee6b2732e8d741daa
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 01:53:04 2017 +0200

    Added file2

commit 4561fce00cf086ae9c3f431e1fa3a38422bcaa26
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 01:52:32 2017 +0200

    Added file1

commit b91ac1dc341511fe7408038f2ed1417d44bf3418
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 01:43:36 2017 +0200

    Added file for test
```

- merge 2 branches

1. Let's say we work in a project with multiple people, the main project is on master branch, however we work on a separate branch and we want to apply the modifications so that everybody will have the updated version of the project. In this case we need to merge our branch with master. We can do this by going to master branch and using:

```
git merge branch1
```

We need to make sure we are on the branch we want to merge with.



1. create a meaningful pull request

1. First of all, let's talk for what we will use a pull request. In our case we will make an update to the ide labs repository and we'll make a pull request so that the ones with access to repository will decide if they want to update or no.
2. First step is to fork the repository. To do this we go to github repository we want and we press the fork button. By doing this we will have now the exact copy of the repository on our account, and we can make changes to it.
3. We clone the repository on our local machine.
4. We make all the changes we want, in my case I change 2 files.

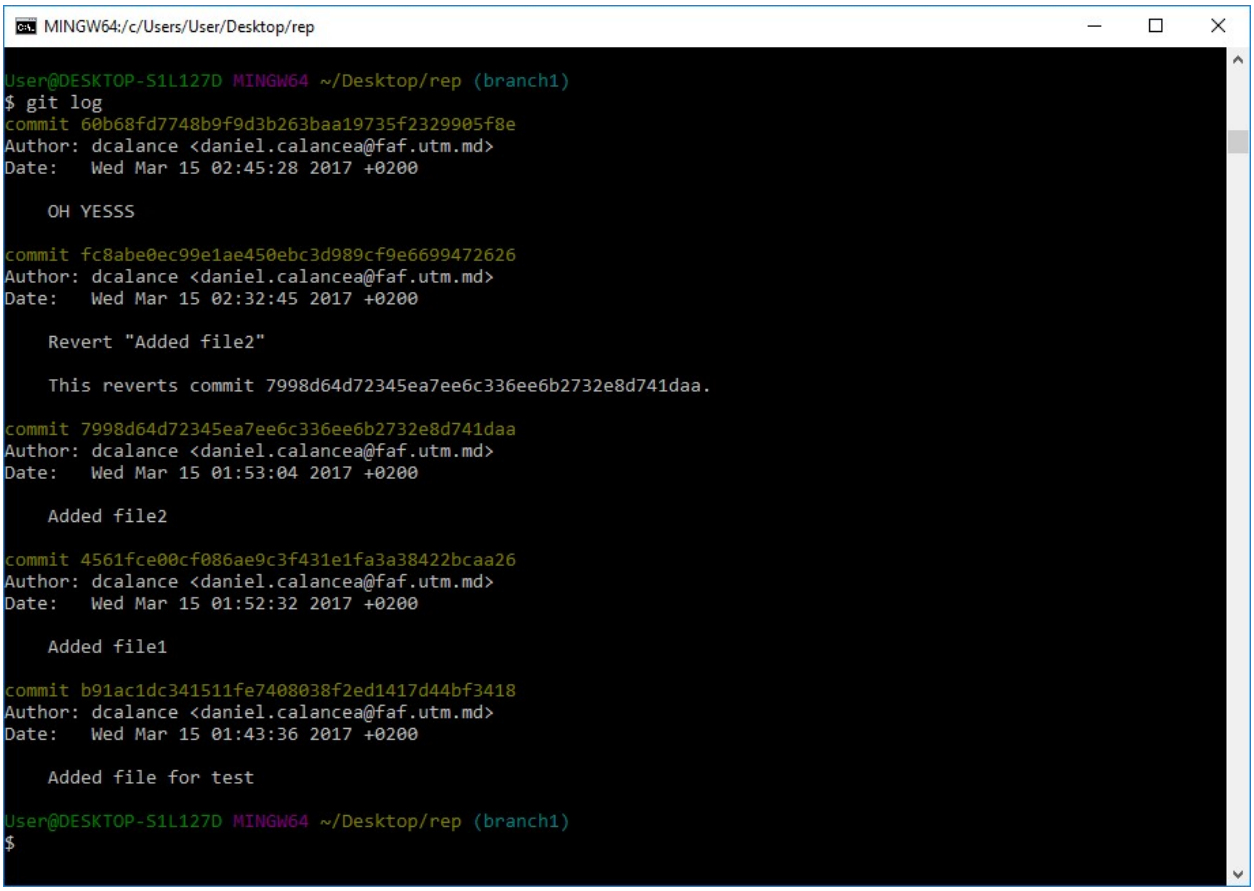
```
MINGW64:/c/Users/User/Desktop/ide-1
User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms/c (master)
$ vim hello.c

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms/c (master)
$ cd ..

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms (master)
$ cd ..

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1 (master)
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
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:   HelloWorldPrograms/c/hello.c

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

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1 (master)
$ git add .

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1 (master)
$ cd HelloWorldPrograms

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms (master)
$ cd cpp

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms/cpp (master)
$ ls
hello.cpp

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms/cpp (master)
$ vim hellp.cpp

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms/cpp (master)
$ vim hello.cpp

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms/cpp (master)
$ cd ..

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1/HelloWorldPrograms (master)
$ cd ..

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1 (master)
$ git add .

User@DESKTOP-S1L127D MINGW64 ~/Desktop/ide-1 (master)
$ git commit -m "Updated c and cpp helloworld programs"
[master 7ece8ac] Updated c and cpp helloworld programs
 2 files changed, 4 insertions(+), 2 deletions(-)
```

5. Now we push all the changes to the remote repository.
6. We go to the github to the forked repository that is located on our account and we press create pull request.

7. In the pull request we can select on what branch we want to create a request. If it is not possible to merge, this means that you have made untracked changes or the branches are not compatible.
8. After that we write the message that will be passed to the developers of the repository you have forked.

- GIT cherry-pick

1. Git cherry-pick is a useful command to apply a diff from a commit to your current file. If we use it we need to specify the commit id, and it will create a new commit in which he will apply the diff from that commit.
2. Let's say we have:



3. We observe the second commit is a revert commit, however we don't want it take effect and we don't want to delete it either, in this case we can use cherry-pick. We will apply cherry pick with the commit that says "Added file2"

```
git cherry-pick 7998d64d72345ea7ee6c336ee6b2732e8d741daa
```

```
MINGW64:/c/Users/User/Desktop/rep
User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git cherry-pick 7998d64d72345ea7ee6c336ee6b2732e8d741daa
[branch1 9b40cb4] Added file2
 Date: Wed Mar 15 01:53:04 2017 +0200
 1 file changed, 1 insertion(+)
 create mode 100644 file2

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git log
commit 9b40cb43b51b3d66d9b479ebc9b5a7be140829e5
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 01:53:04 2017 +0200

    Added file2

commit 60b68fd7748b9f9d3b263baa19735f2329905f8e
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 02:45:28 2017 +0200

    OH YESSS

commit fc8abe0ec99e1ae450ebc3d989cf9e6699472626
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 02:32:45 2017 +0200

    Revert "Added file2"

    This reverts commit 7998d64d72345ea7ee6c336ee6b2732e8d741daa.

commit 7998d64d72345ea7ee6c336ee6b2732e8d741daa
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 01:53:04 2017 +0200

    Added file2

commit 4561fce00cf086ae9c3f431e1fa3a38422bcaa26
Author: dcalance <daniel.calancea@faf.utm.md>
Date:   Wed Mar 15 01:52:32 2017 +0200
```

We can observe that it created a new commit identical to the one we created before. In this example we don't have much use of this feature however it shows exactly what this tool can do.

- GIT hooks

1. Git hooks is a very interesting feature implemented in git. It allows us to execute a script whenever an action happens. We can write scripts in different languages like python, unix shell. The scripts are located in the folder .git/hooks/. There are some sample scripts in there and we can activate them by making it executable (removing the .sample extension).
2. We will make a simple script that will add to every commit message the name and the mail of the person who commited.
3. To do this we will change extension of file **prepare-commit-message.sample** to **prepare-commit-message** and add the code:

```
#!/bin/sh
#
# Automatically adds branch name and branch description to every commit message.
#
NAME=$(git branch | grep '*' | sed 's/* //')
DESCRIPTION=$(git config branch."$NAME".description)

echo "$NAME"': '$(cat "$1") > "$1"
if [ -n "$DESCRIPTION" ]
then
  echo "" >> "$1"
  echo $DESCRIPTION >> "$1"
fi
```

This is a unix shell script.

4. Now we need to make sure that this file is executable so we will use

```
chmod +x prepare-commit-message
```

5. Now we make a simple commit and observe the changes.

```
MINGW64:/c/Users/User/Desktop/rep

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git commit

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git commit -m "Hello"
Commit successfull
[branch1 0838ac9] branch1: Hello Signed-off-by: dcalance <daniel.calancea@faf.utm.md>
 1 file changed, 0 insertions(+), 0 deletions(-)
 delete mode 100644 file3

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ ls
dweweq  file  file1  file2  yeeeeeessssss

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ touch file3

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git add .

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$ git commit -m "Added file3"
Commit successfull
[branch1 ca24b6d] branch1: Added file3 Signed-off-by: dcalance <daniel.calancea@faf.utm.md>
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file3

User@DESKTOP-S1L127D MINGW64 ~/Desktop/rep (branch1)
$
```

We can observe that name and email were added to our string.

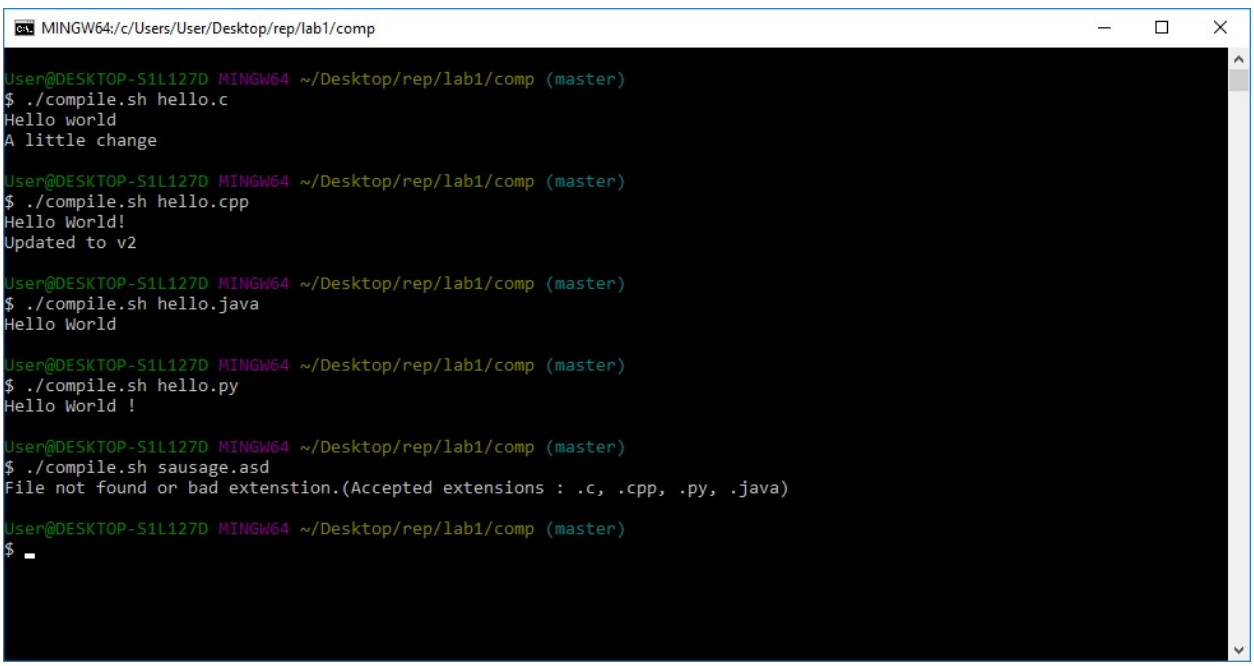
- Write a script that will compile any chosen project from the list of HelloWorldPrograms projects. Make your script output compilation results for each project

1. We will make this script in bash since we can use directly commands from terminal.
2. Our script checks for extension and if the file actually exists for every case. If those conditions are satisfied, then it compiles/interprets. The code is

```
#!/bin/bash
FILE=$1
filename=${FILE%.*}
if [ "${FILE: -2}" == ".c" ] && [ -f $FILE ]
        then
        gcc ${FILE} -o ${filename}
        ./$filename
elif [ "${FILE: -4}" == ".cpp" ] && [ -f $FILE ]
        then
        g++ -o ${filename} ${FILE}
        ./$filename
```

```
elif [ "${FILE: -3}" == ".py" ] && [ -f $FILE ]
        then
        python ${FILE}
#The class containing main function must have name Main
elif [ "${FILE: -5}" == ".java" ] && [ -f $FILE ]
        then
        javac ${FILE}
        java "Main"
else
        echo "File not found or bad extenstion.(Accepted extensions : .c, .cpp, .py, .java)"
fi
```

3. So in order to be able to compile/interpret a file you need to match the extension and in java case you need the class that contains your main function to be named Main, otherwise the program will output a message.



Conclusion:

In this laboratory work we learned how to connect to another machine using a RSA encryption and SSH key. We also saw how many functions have a repository and why all companies are using them nowadays in their everyday work. Shell scripts are useful when you use bash commands however the syntax is horrible and pretty hard to understand. We can create a repository now, link it to a remote repository, commit, manipulate commits, push to the repository, modify and make pull requests for another repositories.