IE3030 Linux – Laboratorio 3 GIT

Por: Ing. Jonathan de los Santos

I. MATERIALES

TABLA I
MATERIALES

Cantidad	Instrumento o componente
1	Computadora con cualquier distro de Linux instalada físicamente o virtualizada.

II. OBJECTIVES

1. Aprender los conceptos básicos de Git.
2. Usar Git para tener un backup de los proyectos realizados.

Nota 1: todos los créditos de este laboratorio son para el autor del mismo "University of the Pacific". https://ecs-network.serv.pacific.edu/ecpe-170/lab/lab-version-control únicamente se adaptó a las necesidades de UVG.
Nota 2: use en todo momento el curso de Cisco NetAcad como soporte teórico. https://www.netacad.com/courses/os-it/ndg-linux-unhatched

III. EXPERIMENT

A. *PART 0: Creating a Repository*

Create an account at BitBucket.org, a free service that can host Git repositories. BitBucket is a "freemium" service. You can have an unlimited number of private repositories in your account, but you can only share your repositories with a limited number of users. Thus, the free service is suitable only for small groups, but is sufficient for typical class projects.

1. Go to the www.BitBucket.org website
2. Click the Signup button
3. Enter your name, email, and password to create a new account

Next, create a private repository (i.e. a version control folder) in your account. This repository will hold all your your course work for this class, including: C source code, MIPS assembly code, Python code, PDF documents, plain text files, scripts, etc...) Inside the repository, you can create as many nested folders as you want to organize things. Although private, you will grant me read-only access to your repository so that I can grade your work.

1. Choose **Repositories->Create Repositories**.
2. Enter the project name: **Project1** (or whatever you want).
3. Enter this repository name: **UVGIE303020221** (Why did I choose this name? I hope you'll use this tool for other courses in future semesters / years, and wanted an organized naming scheme).
4. Make sure the private box is checked - your work is not to be shared by default.
5. Default branch name: main
6. Click Advanced Settings
7. Select the language type: e.g Shell.
8. Enter a description: (whatever you want)
9. Select Create Repository

Finally, add a special instructor account as a read-only user on the private repository you just created. For grading, I will take the last commit into your repository that was submitted before the posted deadline. You must submit all your work in Canvas as well, never forget it!

1. While on your repository web page, select Repository Settings on the left side
2. Select User and Group Access on the left side
3. Under the Users area, enter the email: jadelossantos@uvg.edu.gt. Click Add.
4. Select the Read button to invite this instructor account to have read-only access. Notify any issues if exist.

Checkpoint to obtain credit:
Submit to the Canvas assignment
1. the email address used to signup with BitBucket, and
2. the "git clone" command for your repository.

You can find it by clicking the "Clone" button in the upper right corner and then changing from "SSH" to "HTTPS" in the dropdown.

This command should look like this: git clone git clone https://jadelossantos@bitbucket.org/jadelossantos/uvgie303020221.git

I will use this command to get an initial (empty) copy of your coursework repository, and pull the latest updates throughout the semester for grading.
I will also grant you read-only access to a repository containing initial code for future labs.

B. *PART 1: Using your Personal Repository*

Assuming that you have already created your repository at BitBucket (or at one of any number of other online services), the first thing to do is make a copy of the repository on your local computer. This action -- making a complete copy of a repository -- is called cloning.

When cloning a repository, you need to determine two things:

- What is the repository I want to clone, and where is it stored?
  (The git clone command performed in the PART 0 of this lab contains the answer to this question)
- Where do I want to place the repository on my local computer?
  (My personal preference is a folder in my home directory with an obvious name, so I know anything inside it is under version control)

If you didn't install Git when setting up your virtual machine, install it now.

```
unix>  sudo apt-get install git
```

First, make sure you are at your home directory. Then, create a directory to hold all of your repositories, and then enter it.

```
unix>  cd
unix>  mkdir bitbucket
unix>  cd bitbucket
```

Now, inside the nice obvious "bitbucket" directory, place a clone of your repository. (Make sure to replace <yourname> with your actual username):

```
unix>  git clone
https://jadelossantos@bitbucket.org/jadelossantos/uvgie303020221.git
http authorization required
realm: Bitbucket.org HTTP
user: yourname
password:
destination directory: UVGIE303020221
no changes found
updating to branch default
0 files updated, 0 files merged, 0 files removed, 0 files
unresolved
```

Think of this cloned repository as your own **private sandbox**. It has a complete copy of all files in your project. You can add, modify, and delete files here to your heart's content, and it won't affect the online repository until you explicitly upload your changes.
Now, enter the folder for your new repository:

```
unix>  cd UVGIE303020221
```

To successfully commit your files, you need to let Git know the username and email associated with it. Perform the following:

```
git config --global user.email "youremail@uvg.edu.gt"
git config --global user.name "Your Name"
```

For example, on Dr. Pallipuram's computer, configuration (git config) would look something like:

git config --global user.email "vpallipuramkrishnamani@pacific.edu"

git config --global user.name "Vivek Pallipuram"

Inside the repository, let's create a directory structure to keep organized.

```
unix>  mkdir lab{02..12..1}
unix>  mkdir exam{1..2..1}
```

This uses "brace expansion" in the Bash shell to run a for-loop and create many directories with the name lab02, lab03, lab04, etc.. It is the equivalent of this longer command: mkdir lab02 lab03 lab04 lab05 lab06 lab07 lab08 lab09 lab10 lab11 lab12 exam 1 exam 2

More about brace expansion: https://www.howtogeek.com/725657/how-to-use-brace-expansion-in-linuxs-bash-shell/

You will use this one repository for all of your work in IE3030.

Whenever you work on a lab, be sure you are working inside the corresponding repository lab folder created above.

For other classes, create separate repositories to stay organized.

By design, Git won't track empty folders. To bypass this -- and to eliminate any possibility of grading confusion -- we're going to put a text file named author containing your name in each folder. The echo command (as the name implies) will output the string in quotes on standard output. The caret (>) redirects the output to the specified file, which will be created if it doesn't exist. Repeat once per folder.

```
unix> echo "YOUR NAME" > lab02/author
unix> echo "YOUR NAME" > lab03/author
unix> echo "YOUR NAME" > lab04/author
unix> echo "YOUR NAME" > lab05/author
unix> echo "YOUR NAME" > lab06/author
unix> echo "YOUR NAME" > lab07/author
unix> echo "YOUR NAME" > lab08/author
unix> echo "YOUR NAME" > lab09/author
unix> echo "YOUR NAME" > lab10/author
unix> echo "YOUR NAME" > lab11/author
unix> echo "YOUR NAME" > lab12/author
unix> echo "YOUR NAME" > exam1/author
unix> echo "YOUR NAME" > exam2/author
```

At this point, you have a simple directory structure and some files in your repository.

Go to the BitBucket.org website and log in. Select your repository. View your repository files by clicking on the Source tab (< >). **Is anything there?**

Lab Report:

(1) Take a Screenshot (using the Ubuntu Screenshot tool) showing what the BitBucket source view looks like at this point.

By design, you have to explicitly tell Git when to save a version, and when to copy it from your local repository to a remote repository, such as the BitBucket site. Until you do that, there will be nothing shown at the BitBucket website. Let's start that process now.

It is a good practice to commit work to the version control system frequently - whenever you are at a convenient stopping point.
First, check and see the status of your current repository:

```
unix>  git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    lab02/
    lab03/
    lab04/
    lab05/
    lab06/
    lab07/
    lab08/
    lab09/
    lab10/
    lab11/
    lab12/

nothing added to commit but untracked files present (use "git add" to track)
```

All those "Untracked" files indicate **files that are not currently tracked by the version control system**. You need to explicitly tell Git to track a file by using the **add** command - otherwise its history will not be recorded. Let's add all of the "author" files now:

```
unix>  git add lab*/author exam*/author
```

Notice the use of the wildcard (*) above.

Re-run the git status command now, and note the difference:

```
unix>  git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   lab02/author
    new file:   lab03/author
    new file:   lab04/author
    new file:   lab05/author
    new file:   lab06/author
    new file:   lab07/author
    new file:   lab08/author
    new file:   lab09/author
    new file:   lab10/author
    new file:   lab11/author
    new file:   lab12/author
```

Instead of a "Untracked" indicating an untracked file, you get an "new file" indicating a newly-added file.

To know more about the status command, you can always find out from the built-in Subversion help system:

```
unix>  git help status | less
```

Simply adding a file is not enough. Git won't track every keystroke you make. Rather, it will save snapshots whenever you explicitly tell it to via the **commit** command. It should be apparent from the above output where Git is telling you "No commits yet". Let's do that now:

```
unix>  git commit -m "Creating initial directory structure for class repository"
```

The commit command saves a snapshot of all files that Git is tracking (via prior use of the add command). This snapshot is saved to your *local* repository. Notice my use of the optional -m argument to specific the **commit message**. This message appears in the log of all commits made to the repository. Imagine you've been working on a large project for several weeks, and have dozens of commits. You want to look at the code right before you made a major change to the project. **Having descriptive commit messages makes it easier to find the right place in the project history**.
Re-run the git status command now, and note that the newly added files no longer appear. This is because you've told Git to save a snapshot of their contents, and they have not been modified since the last snapshot.

```
unix>  git status
```

Go to the BitBucket.org website again. View your repository files by clicking on the Source tab. **Is anything there?**

Lab Report:

(2) Take a Screenshot (using the Ubuntu Screenshot tool) showing what the BitBucket source view looks like at this point.

Git is what is known as a **Distributed** Version Control System. Essentially, the repository on your local disk is an independent entity from the repository on the BitBucket server. In order to have a good backup, you need to **push** from your repository to the BitBucket repository.

```
unix>   git push
Password for 'https://yourname@bitbucket.org':
Counting objects: 4, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (4/4), 319 bytes | 159.00 KiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To
https://bitbucket.org/vivek_pallipuram/2021_fall_ecpe170.git
 * [new branch]      master -> master
```

Go to the BitBucket.org website again. View your repository files by clicking on the Source tab. **Is anything there?**
There should be!

Lab Report:
(3) Take a Screenshot showing what the BitBucket source view looks like at this point.
(4) Take a Screenshot showing what the BitBucket version history looks like at this point (under the Commits tab)

Let's continue working in our repository. Enter the lab02 folder:

```
unix>  cd lab02
```

Create a text file called reasons_to_use_vcs.txt and open it in an editor:

```
unix>  gedit reasons_to_use_vcs.txt &
```

Go **read the discussion** on version control systems at http://stackoverflow.com/questions/1408450/why-should-i-use-version-control.
In your text file, briefly summarize three reasons to use version control systems in the form of a bulleted list.
This is a good point to add the new file to Git, and check it in:

```
unix>  git add reasons_to_use_vcs.txt
unix>  git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   reasons_to_use_vcs.txt
unix>  git commit -m "Lab 2: Added file on why version control is awesome"
unix>  git push
```

Typically I always run the git status command before an git commit. Why? **The status command will show you all of the new/modified/deleted files that will be included as part of the commit.** Further, it shows you **any untracked files that you forget to include (by mistake).**

Lab Report:
(5) Take a Screenshot showing what the BitBucket version history looks like at this point (under the Commits tab)

Let's see how Git handles existing files that are modified.

Open the text file and add two more reasons to use version control to your bulleted list. Save the file, and exit.
What is the status of the repository now?

```
unix>  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:   reasons_to_use_vcs.txt

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

The above message tells you that a file has been modified and also kind of tells you what to do next (see the last line).

The Git **diff** command can show you the **difference between the current version of a file, and the file as last committed to the repository**. In other words, what has changed?

```
git diff
```

The output of this command is in "diff" (difference) format, which is a common Unix utility. Let's see if you can deduce what each line represents.

Lab Report:
(6) What does the line starting in --- represent?
(7) What does the line starting in +++ represent?
(8) What do the lines starting in + represent?
(9) Can you guess what a line starting in - would represent? (Depending on the edits you made, you might have a few of these lines too)

Tip:
https://jaimeiniesta.github.io/learn.github.com/p/diff.html

Now commit your updated work:

```
unix>  git commit -a -m "Lab 2: Added additional reasons on why version control is awesome"
unix>  git push
```

Note the above commit line. The -a tells the commit command to automatically stage files that have been modified and deleted, but new files you have not told Git about are not affected.
Version control systems (like Git) provide special commands to copy or move files. You should use these special commands, instead of the generic Unix cp or mv commands, because they **preserve version history.** (Thus, for example, if you rename a file, you could see its history back through its original filename).
To move a file within a Git repository, use: git mv <filename> <destination_filename>
There are more topics that we haven't covered in this introductory tutorial. For example:

- **Branching**, **Merging**, and resolving **Conflicts** - In this scenario, different users modify the same files at the same time, committing them to their own private repositories, and then trying to combine their work together at a later date. That is really what these tools are designed to help with!
- **Reverting** to an earlier version of a file (or project)

Lab Report:
(10) What benefits would a large team of developers get from version control? Identify at least two.
(11) What benefits would a single developer (working alone) get from version control? Identify at least two.
(12) What kind of files should you put in version control?
(13) What kind of files should you **not** put in version control? Why?
(14) What is the difference between an add, commit and a push?
(15) Why is it better to use git mv instead of plain-old regular mv to copy a file within the repository?
(16) How would you suggest improving this lab in future semesters?

Lab Report Submission: Check this lab report into your BitBucket.org personal repository (under the lab02 folder) by the posted deadline. Reports must be submitted as PDF files, not as LibreOffice / MS Office files (Choose File->Export as PDF).