SECTION 05

Version control using Git

THE NEED FOR VERSION CONTROL

- As DevOps is all about speed, efficiency, and collaboration, there should be a version control system in place. Why?
- Version controlling your code provides the following advantages:
    - The ability to work on the same project by more than one person at the same time. Each person will have a copy of the source code to do whatever changes required, then changes are *merged* with the main code, after the necessary approvals.
    - Protection against errors and failures by enabling developers to go to and from different versions of the same code. Debugging is greatly facilitated this way.
    - Code can be *branched*. Developers can work on separate branches of the source code to create and test new features. At the end a branch can be merged with the main branch (called master) or deleted.

IS IT GIT OR GITHUB?

- Due to its huge popularity, some people confuse GitHub for Git. Actually, they are two different things.
- Git is the program used to version control your files. It was created by Linus Torvalds (the creator of the Linux kernel) in 2005. It was to version control Linux kernel 2.6.12.
- GitHub, on the other hand, is a web application that was launched in 2008. It offers a version control repository that can be used free of charge or for a fee (for private repositories).
- Git can be integrated with GitHub so that the repository can be accessed over the Internet from anywhere in the world instead of having to create a local repository server.
- GitHub is not the only Git repository in the market; as there are similar products like BitBucket.

GIT INSTALLATION ON MAC AND WINDOWS

- Most Linux distributions come pre-bundled with Git. You can check that by running `git --version`.
- If you are on a macOS (version 10.9 or higher), you will have to install Xcode command line tools before attempting to install Git. Typing git on the command line will prompt you for doing that.
- Alternatively, you can download the package and install it manually by visiting https://git-scm.com/download/mac or https://git-scm.com/download/win
- You can also install GitHub desktop by visiting this URL: https://desktop.github.com
- There are a number of GUI versions for Git that can be downloaded from this URL: https://git-scm.com/downloads/guis

GIT INSTALLATION ON LINUX

- The easy method of installation is just using the package manager. For example: `yum -y install git` for rpm-based machines (Red Hat, Centos, Fedora…ect.) or `sudo apt-get install git` for Debian ones.
- The problem is, this may not necessarily grab the latest version of Git. That’s because vendors must do massive testing on packages to ensure full compatibility with their systems. This leads to a version that is lagging behind the current released one.
- So, if you are desperate to get the latest version of Git, you can do that by compiling it from source. Let's see how.

LAB: COMPILING GIT FROM SOURCE - UBUNTU INSTRUCTIONS

- First, we'll use our already-installed git command to grab the latest version of the product. Issue the following command `git clone https://git.kernel.org/pub/scm/git/git.git`
- Second, we need to install some prerequisites for the build process to work. Issue the following command: `sudo apt-get install libcurl4-gnutls-dev libexpat1-dev gettext libz-dev libssl-dev asciidoc xmlto docbook2x`
- Now, let's start the actual build process. Issue the following commands:
  make all doc info prefix=/usr
  sudo make install install-doc install-html install-info install-man prefix=/usr
- It may take up to a few minutes to compile. Once done, run the following command to check the version: `git --version`.

LAB: COMPILING GIT FROM SOURCE - CENTOS INSTRUCTIONS

- Same to Ubuntu, we need to install some prerequisites (you can use Ansible for that):
```
sudo yum -y install epel-release
sudo yum -y groupinstall development
sudo yum install curl-devel expat-devel gettext-devel openssl-devel
perl-devel zlib-devel asciidoc xmlto docbook2X
```
- Now, we need to make a symbolic link: `sudo ln -s /usr/bin/db2x_docbook2texi /usr/bin/docbook2x-texi`
- Clone the latest git repository from kernel.org: `git clone https://git.kernel.org/pub/scm/git/git.git`
- Compile Git by running the following commands from inside the cloned git directory:
```
make all doc prefix=/usr && sudo make install install-doc install-
html install-man prefix=/usr
```
- Again, issue `git --version` to ensure that you have a successful installation.

YOUR FIRST GIT STEPS

- The first thing you can do is to - optionally - tell git about your name and e-mail.
- This is useful when you want to login to remote repositories, issue pull and merge requests and for other purposes.
- To save your details issue the following commands:
  git config --global user.name "Ahmed El-Fakharany"
  git config --global user.email "abohmeed@gmail.com"
- The second thing you can do is initialize your repository. Initialization refers to informing Git about a directory that contains the files you need to put under source control.
- In our project, files are located in /var/www/html/CodeIgniter-3.1.5/. Navigate to that path and issue the following command: git init.
- Now issue `ls -la` to show hidden files. You'll find a new directory created called `.git`. Inside it there are a number of files that help Git do its job.

COMMITTING YOUR CHANGES

- Git works by watching a directory for changes in the files inside. Creation, deletion, or modification actions are all tracked.
- However, Git does not do that automatically. You have to inform it about the files and directories it needs to watch. This is done by using `add` subcommand.
- While inside our project directory, issue the following command: `git add *`
- This command will instruct Git to watch all files and directories in the current location for changes.
- So far, you are in the “staging” state. The files and directories have been considered by Git, but not saved yet.
- To save this content, you have to issue the commit subcommand. Issue git commit
- As soon as you issue this command, a vi editor (or the configured editor) will automatically launch. It has a pre-written document containing the actions that were made. In our case, a lot of new files added.
- This message helps you and your fellow colleagues determine what exactly happened at this point in time.
- Add the following to the end of the file: `"Intitial commit"`
- Save the file and the you have your first commit.
- You can (and probably will) add this message by supplying the -m option followed by your message on the command line. For example, `git commit -m "Initial Commit”`

REVIEWING THE CHANGES

- Whenever you want to see the previous commits, issue `git log` command.
- The output will contain:
  - The hash of the commit: this is the unique identifier that differentiates different commits. You can later roll back to this commit using this identifier.
  - The author: that's the name you specified earlier. The e-mail is also displayed.
  - The date of the commit
  - The message of the commit.

REVIEWING THE CHANGES

- Git manages file versions. You can have different states of the same file and go forward and backward through them.
- Let’s change the database IP address in the CodeIgniter configuration, commit that change and see how can we know what the file looked like before.
- Change the IP address of the database host in application/config/database.php to any other value.
- Issue `git add *` (or you can be more specific `git add application/config/database.php`)
- Commit the changes with a message: `git commit -m "Changed the database host to be localhost"`
- Now the application may stop working as you are referring to a database engine that does not exist. In your attempt to troubleshoot the issue, you use `git log` to see the changes that happened.

- You need to know what changes happened in the file. So, you copy both commit hashes (the one from the latest commit and the previous one) and issue the git diff command as follows: `git diff 7a26f5b53e00a08d53d33af799c4a8b32ce18d9d..9cea8c92bcbf92c034ca61a1e0be79bb4bc994e9`
- The output of the above command shows that an author has changed the database host in application/config/database.php to be something wrong.

WORKING WITH REMOTE REPOSITORIES

- So far, we've been working on the local machine. This may be sufficient if you are working alone, or if all of the team is working on the same machine.
- But in real-world scenarios, this is highly unlikely. People work on projects on their personal laptops. They may not even be physically in the same country or region.
- For this reason, Git offers the capability of working with a central repository server. Any server can act as a central repository, or you can use a service like GitHub.
- Create a free account on www.github.com. It's free and only requires a valid e-mail address.
- Once you have an account, create a new project (repository). I'll call this project "swift".
- Let's add all our content now to GitHub under the swift project.

CLONING AND PUSHING

- Issue the following command:
  ```
  git remote add origin https://www.github.com/abohmeed/swift
  ```
- “Origin” is the name we chose for the remote. This could be a name of your choice.
- To actually start adding your files and directories to the remote Git repository, issue the following command:
  ```
  git push -u origin master
  ```
- “Master” is the name of the local branch. You’re basically “pushing” files from master to the remote origin branch.
- You’ll be asked for your login credentials and then the files and directories will be uploaded. Actually Git converts the files to binary format, subsequent push requests transfers only the changed parts of the files. This ensures faster operations.
- As you progress through your project and make many changes that needs to be pushed periodically to the remote branch, you may want to set it as the default destination. This can be done by issuing the following command:
  ```
  git push --set-upstream origin master
  ```
- Now, whenever you need to push changes to the remote repo, just issue `git push`

- We now have a remote repository populated with our files. Anybody can clone it to their own machines and start working by issue a git clone command. For example, `git clone https://www.github.com/abohmeed/swift`
- Let's clone this repo on our client machine in /vagrant so that we can work on it from the host using a code editor.
- The database configuration file is still pointing at a wrong location. We need to fix that. Change the database host to point to the correct address, 192.168.33.30 in our case.
- Commit the changes and push them to the remote repository by issuing `git push`
- You can have a look at the repository on www.github.com and ensure that the file is updated.
- Now the developer working on the application server needs to continue working with the project. He is aware that changes were made to the repository so he needs to have the latest version of the code.
- One option is to delete the directory with all its contents and issue git clone but will take time.
- The better option is to use git pull.

PULLING CHANGES

- On the web server, issue `git pull`
- Since the repository is already configured to track changes on the remote host on GitHub, any changed files will automatically be synced locally.
- But, maybe this developer was intending to keep the database pointing at localhost for some reason, perhaps testing a new database engine that yet to be installed.
- In this case, git offers you the `fetch` subcommand. Issuing git fetch will just bring any changes done remotely, but without actually committing them.
- On the client machine, change the welcome message on application/views/welcome_message.php to something else. Commit the change with a descriptive message and push to the remote repo.
- You can view the changes done and then issue `git merge` or `git pull` to actually commit them.
- But how can you determine what has changed?

DETERMINE THE CHANGES BROUGHT BY GIT FETCH

- After issuing git fetch, if you issue git log you won't see any new commits. Just the same last commit entry.
- To view the changes, simply issue `git log master..origin/master`. This will list the changes done on the remote branch (origin/master) but not yet committed to the local branch master.
- There you go, there is a change that reads "changed the welcome message". If you are not sure that this change actually did, you can grab the commit id and find the difference between it and the last commit id, that was created on your machine: `git diff 997b12708f7f766b898823df4046e78d7325f6eb.. 1b464c9f63aefeb27694980c97bfe755757a1cc0`
- If you want to apply those changes, just issue `git merge` or `git pull`

BRANCHING

- Branching is one of the very reasons why version control systems exist. Let' say that a group of developers disliked the layout of the web application. They decided to change it by modifying the HTML and CSS files.
- Such a change must be thoroughly tested and approved before finding its way to production. Before version control, the typical method was to take a copy of the source code, work on it and then manually sync the changed files with the production code. But that is highly error prone.
- Using Git, the developers can create a separate “branch”, called may be “newdesign” and do all there changes there. In the meantime, another group of developers may be working on the JavaScript files to let some pages use AJAX (a cool web page feature).
- Both teams can work simultaneously on separate branches. Once approved, changes can be merged to the master branch. Let’s see how can this benefit our project.

CHANGING THE FONT COLOR

- We need to change the font color of the welcome page. We need a fast way to switch from the original font color of the page to the new one so that we can decide which is better.
- Let's create a new branch called "blue" for that. Issue the following command: `git branch blue` to create the new branch.
- Now, we have two branches master and blue. To know which branch is your current, issue `git branch`.
- To switch to the new branch, we issue `git checkout blue`
- Make the required changes to the font color by editing the welcome_message.php to have the h1 color #000DA2 and view the results. Commit your changes.
- Have a look at the page in the browser. Let's see how the old page looked. This can done quickly be issuing `git checkout master` and refresh the page.

REFLECTING BRANCH CHANGES TO THE REMOTE REPO

- The remote origin does not have any idea we have a new branch in our project. We need to upload this branch to that other developers can pull it and start working on it.
- You can do this by using the -u option of the push subcommand, and specifying the name you want to use with the remote branch. For example: `git push -u origin blue`
- Now, you can just issue pull and push requests from the blue branch and they will be automatically directed to origin/blue.

MERGING BRANCH CHANGES

- Once the changes done on the branch is tested and approved. They can be merged to the master branch.
- This can be done by simply issuing git merge blue while on the master branch to make the new font color available to production.
- Once changes are merged, you may or may not want to continue having the blue branch in the project. If that branch was created for just that change, it can be deleted using the following command: `git branch --delete blue`
- You may also need to remove it from the remote repository. This can be done using `git push --delete origin blue`

ROLLING BACK YOUR CHANGES

- Git is not just about having several versions of your files or working simultaneously on the same code in several branches. It also allows you to revert back to a previous state of your files.
- This can be done using the `reset` subcommand. For example, let's delete some HTML from the welcome_message.php and commit the changes. Perhaps the developer who made this change forgot to refresh the page and ensure that nothing broke.
- Now we need to restore the page to its working state. There are two main scenarios here:
  - You did a mistake when editing a file and committed your changes. You need to undo the commit. This can be done by issuing git reset, which will remove the commit and put you back in the state before you committed them. You can now correct the mistake and commit again.
  - You did a mistake when editing the file and committed your changes. You do not know how to restore those changes. You need to bring the files back to the state they were in at some other point in history. This can be done by issuing `git reset --hard` *`commit`*
- Notice that the hard reset option will get rid of any uncommitted changes to the project. So, you need to ensure that your working state is clean by issuing `git status`.

RESTORING A DELETE FILE OR DIRECTORY

- If you deleted a file by mistake and you need to restore it back, you can use Git for that.
- Git watches files and directories for deletion. However, like you used git add to inform Git about added and changed files and directories, you use git rm to inform git about deleted files and directories.
- For example, let's delete readme.rst file from the CodeIgniter root directory. Issue the following command to inform git about this change: `git rm readme.rst`
- Now if you can issue git log to determine the last point where this file existed within the project history and use git reset to restore it as follows: `git reset --hard commit`
- The main problem with this approach, however, is that you lose any changes that you made to the project post the point where the file existed; as this is now the current point of the project history.
- If you deleted a directory just add `-r` to the git rm command.