Quick2Cloud for Cloud Systems

Built on IBM

CLOUD FOUNDRY

Quick2Cloud College - Course 103

Running the Application Locally with a Repository

Creating the local repository

Let us begin this section of our course creating a repository in your q2c-web-app directory where your sample web application code resides. You have already made one small change to the sample application code in **Course 100** without a repository system in place so we will declare what you have right now to be **ver00** which will be the contents of our soon to be created master branch.

At the end of the repository creation, ver00 (master) will be called the codebase that all features and major changes will be birthed off of.

Quick2Cloud **will now generate a script filled with commands that accomplish the following:**

1. Create the code repository
2. Stage all modified files
3. Commit all staged files
4. Create ver01 branch
5. Restore working directory

OK, now let us take this newly created knowledge for a test drive.

Step 0

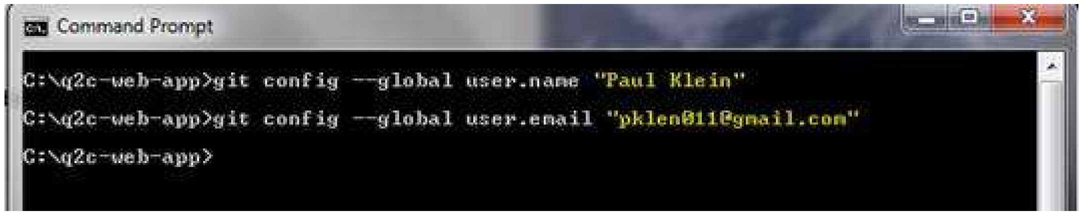
Download the *Quick2Cloud* generated script to accomplish the above. This is a zip file and needs to be **extracted** into the q2c-web-app directory and saved as a .bat or .ps1 file.

Generate: college-git-create

Step 1

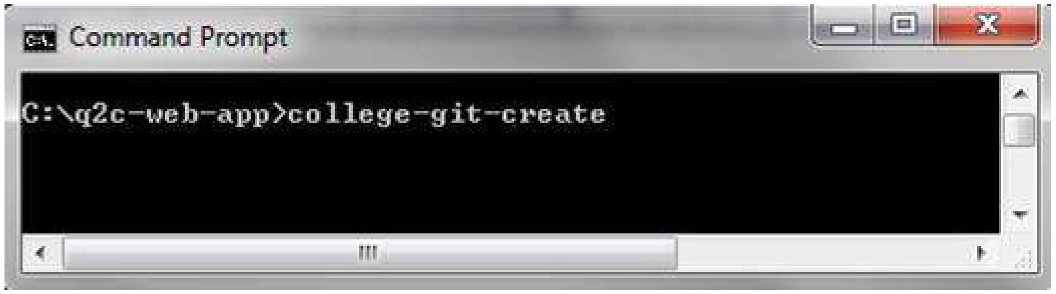
The very first time after installing the git commands you must begin by defining yourself to the git software. This is mandatory or your git commands will not work. Launch the DOS Command Prompt. Change the directory to q2c-web-app and issue the following commands.

1. git config --global user.name "Paul Klein"
2. git config --global user.email "pklein011@gmail.com"



```
C:\q2c-web-app>git config --global user.name "Paul Klein"
C:\q2c-web-app>git config --global user.email "pklen011@gmail.com"
C:\q2c-web-app>
```

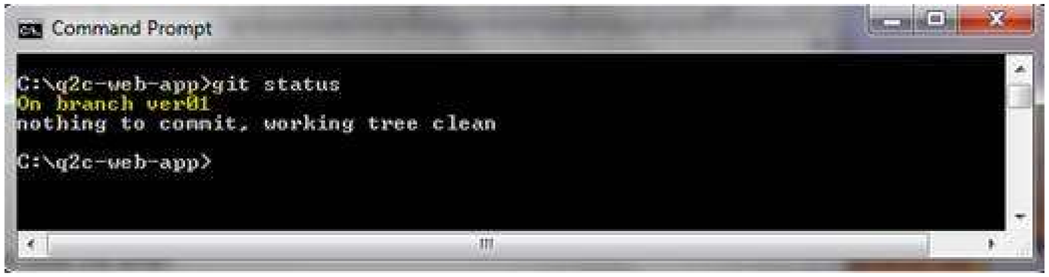
Once this on-time-only user identification is done you can move on to issuing repository scripts. Issue the script below:



```
C:\q2c-web-app>college-git-create
```

Step 2

Let us validate that the repository was created. Using the **git status** command we will see that the repository is there and we are located on the ver01 branch. Since we already staged and committed our working directory as part of the above script, currently there is nothing more to commit.



```
C:\q2c-web-app>git status
On branch ver01
nothing to commit, working tree clean

C:\q2c-web-app>
```

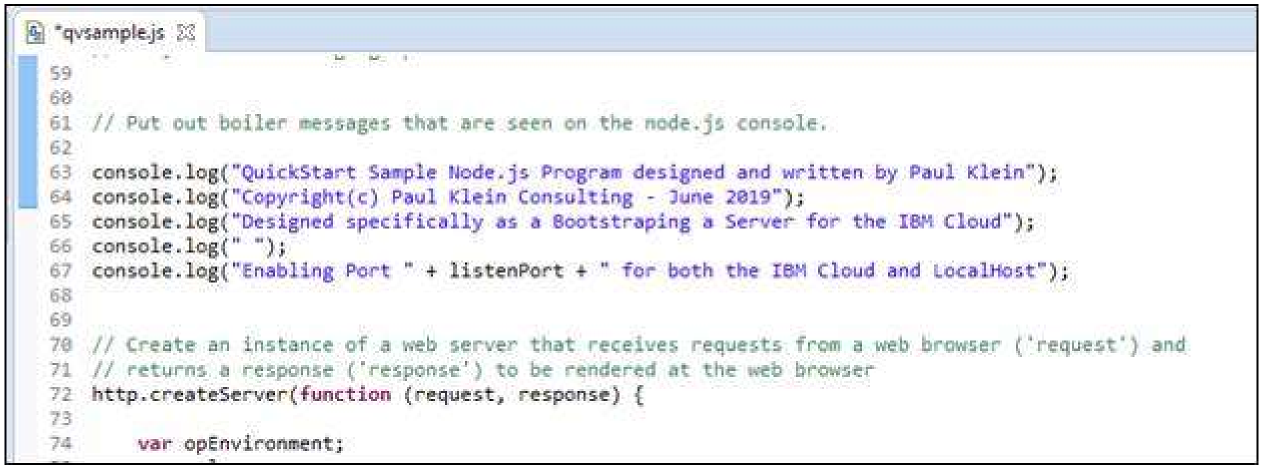
At this point the master and ver01 branches of the repository reflect the same code and they both are also the same as your working directory. As you develop the sample application by committing changes out of the working directory, all these features will be stored under the ver01 branch with the master branch staying the same as it was when your repository script was initially run.

Therefore, we declare with confidence that the master branch, Snapshot 0, houses our ver00 production code with the ver01 branch□s Snapshot 0 housing our latest in progress ver01 feature code and your working directory holds the in progress work. After a number of more committed features - it will be christened ver01 complete**!**. Then you will create another branch, **ver02**, and continue development for that next release.

Note: Once ver01 has been christened complete, you can **push** it into the master branch, continuing master**,** snapshot 0, as production but now containing the latest tested feature code.

Step 3

To test our repository out, you will make a change to the qvsample.js server code then do a git add followed by a commit to the ver01 branch. Fire up your code editor and make the change below.



*qvsample.js

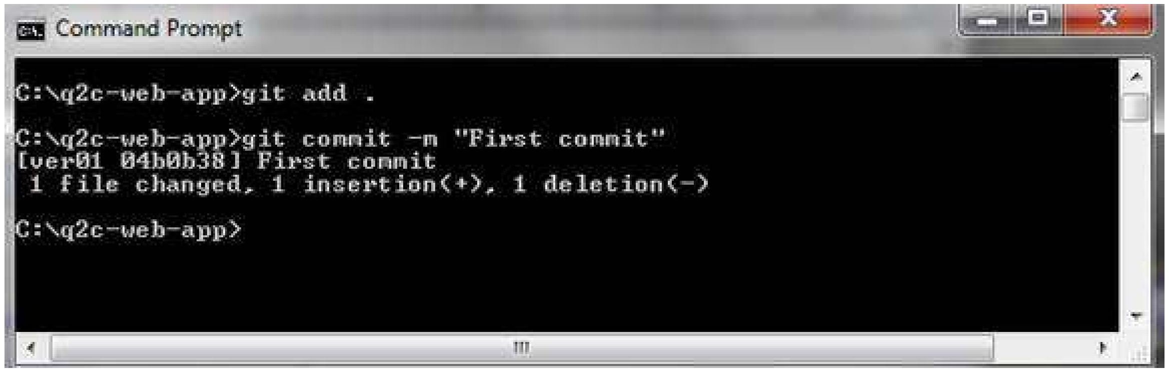
```


// Put out boiler messages that are seen on the node.js console.

console.log("QuickStart Sample Node.js Program designed and written by Paul Klein");
console.log("Copyright(c) Paul Klein Consulting - June 2019");
console.log("Designed specifically as a Bootstraping a Server for the IBM Cloud");
console.log(" ");
console.log("Enabling Port " + listenPort + " for both the IBM Cloud and LocalHost");


// Create an instance of a web server that receives requests from a web browser ('request') and
// returns a response ('response') to be rendered at the web browser
http.createServer(function (request, response) {

    var opEnvironment;
```

Change my name, Paul Klein, on line 69, to John Doe in the console.log() so that the sample program now contains a different name. Then save the file.

Step 4

qvsample.js is now in the **modified** state since it was edited. Let us add it to **stage** in preparation for a **commit**; then issue the commit. You do this by issuing the following:

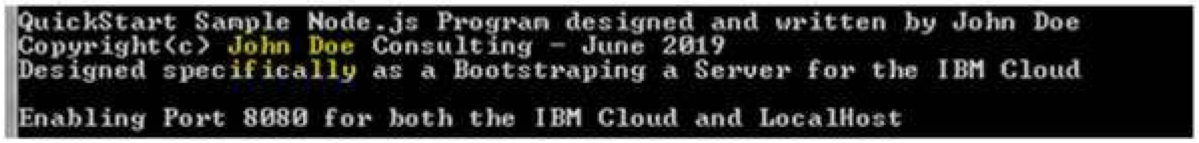
1. git add .
2. git commit -m "First commit"



```
C:\q2c-web-app>git add .

C:\q2c-web-app>git commit -m "First commit"
[ver01 04b0b38] First commit
 1 file changed, 1 insertion(+), 1 deletion(-)

C:\q2c-web-app>
```

Step 6

If we run the sample application, qvsample.js, locally out of the ver01 working directory you will see your name (John Doe) appear in the console as the messages roll up.

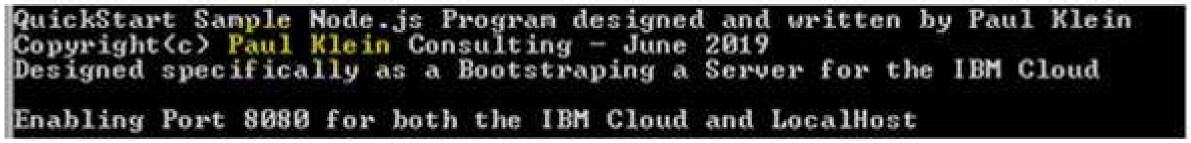


```
QuickStart Sample Node.js Program designed and written by John Doe
Copyright(c) John Doe Consulting - June 2019
Designed specifically as a Bootstraping a Server for the IBM Cloud

Enabling Port 8080 for both the IBM Cloud and LocalHost
```

However, look what happens when we switch to the master branch by issuing the following:

1. git checkout master

When you restart qvsample.js here is what you see:



```
QuickStart Sample Node.js Program designed and written by Paul Klein
Copyright(c) Paul Klein Consulting - June 2019
Designed specifically as a Bootstraping a Server for the IBM Cloud

Enabling Port 8080 for both the IBM Cloud and LocalHost
```

The name rewound back to Paul.

Switching Branches

When you switched to the master branch you first deleted then restored your working directory with ver00 master, snapshot 0. In our scenario that happens to be the code at the time the *Quick2Cloud* repository was initially created. Switch back to the ver01 branch then notice the working directory is deleted again and now restored with the qvsample.js code from your edit in ver01.

Sit back and think about that for a moment. It is one of the more interesting aspects of the repository system. As you switch branches the working directory changes as well.

Note: We are about done on the topic of local repositories and on the way to understanding remote repositories in preparation for **Course 104** and Continuous Delivery. I□m sure you have further questions on repository methodology, read here, but the hope is what has been shown will suffice to get your application on the way to a basic Continuous Delivery enablement.

However, a few comments on some interesting repository workings are worth noting:

FYI

Whenever you are asked to leave what you are doing in one branch and go back to fix something in another branch, you have to either first commit what you have been working on or **stash** it away for later. That is because if you try to switch (**git checkout**) to another branch with modified or staged files outstanding - git checkout refuses the switch.

Why you ask? If it let you switch; you would lose your edits when the working directory was restored to the other branches□ snapshot 0. Think of a stash as a temporary commit that can be undone a short time later.

To stash away your current modified or staged files in your current branch

1. git stash push

Then switch away to another, fix the problem, and then switch back to the original branch

2. git stash pop

This brings back the stashed, modified or staged files. Stash is a push/pop styled function so you can push as many code changes as you want as long as you pop that same amount to back out.

When you switch to a branch you can always switch to the branches prior snapshots by using an index number:

3. git checkout ver01 -1

This would give you the ver01 snapshot that is one back (snapshot -1). You can view the code and even make changes and run tests out of your working directory, however, any new changes won□t stick! Once you switch (git checkout) to another branch, changes you made will be lost forever.

If you could have the past change the future, we would be living Doc Browns□ Back-to-the-Future paradox and you know deep down that would make the universe explode. Instead, make a new branch that is a continuation of the current branch (**ver01-a**) and move forward with changes from there.

You should have enough basic repository knowledge to move forward to the next course but more importantly, you are on your way to mastering one of the Clouds most powerful development methodologies in deploying applications - Continuous Delivery.

Main Menu