Example using Git ISPF client and IBM DBB

Created by rbarosa@us.ibm.com – Reviewed by Liam Doherty July 22, 2020

Introduction

This paper is created to help starting using Git/ISPF, does not intend to replace any IBM documentation, but is as "as is" example.

In this paper I will provide an example using the Mortgage application provided by IBM as sample and using the z/OS running on my ZDT environment.

Git ISPF client provides an ISPF interface that interacts with a Git repository to allow cloning, staging, checking in, pushing and pulling as well as other git commands. Look at the Git/ISPF install and documentation at: https://github.com/IBM/dbb/tree/master/IDE/GitISPFClient

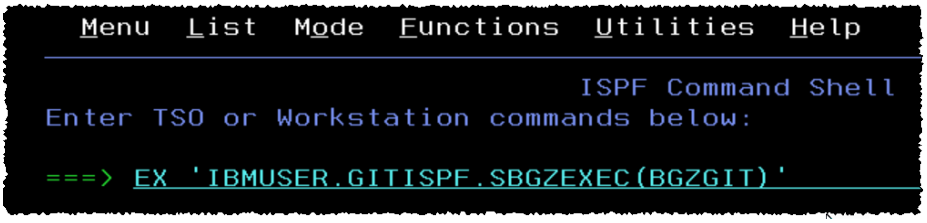
On our example **using the Git ISPF** the main tasks are:

1. Cloned the ***GitHub*** provided Mortgage COBOL application to the z/OS (Rocket Git)
2. Edited a COBOL program using the ISPF editor and changed a comment line.
3. Requested a DBB USER Build to compile and link the modified program using the provided Groovy Scripts by zAppBuild
4. Issued a **Git Commit** and **Push** the changes to the ***GitHub***

Running the ISPF Git Client

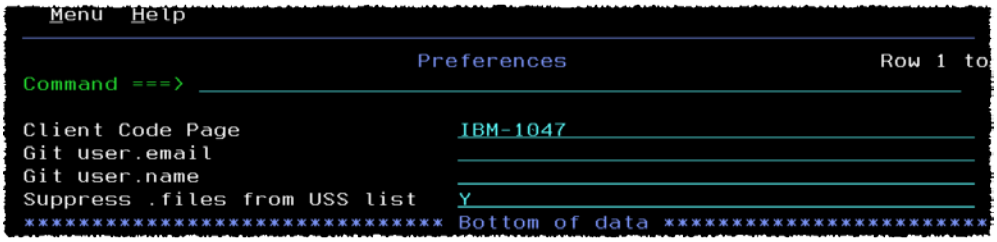
Start the ISPF Git client dialog using the BGZGIT REXX executable code.
On my example:

⏭ (**EX 'IBMUSER.GITISPF.SBGZEXEC(BGZGIT)'**)



⏭ Once you have started the ISPF Git client you should go to Option **0 - Preferences**.

⏭ I made no changes.. **PF3** to return.



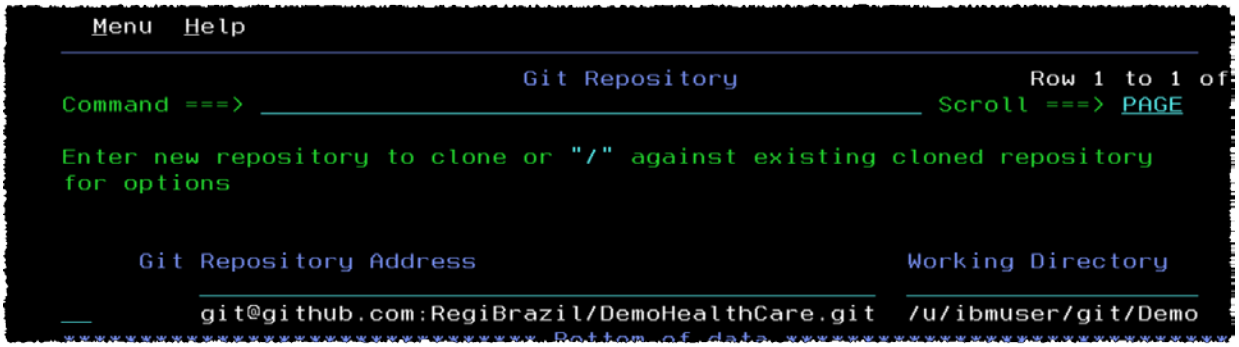
1. Cloning a remote repository

To start working on code in a remote Git repository

1.1. ⏭ Go to option **2 - Repository**.

You will need to enter a repository address in the line at the top of the screen.

In my example I have already a repository cloned:

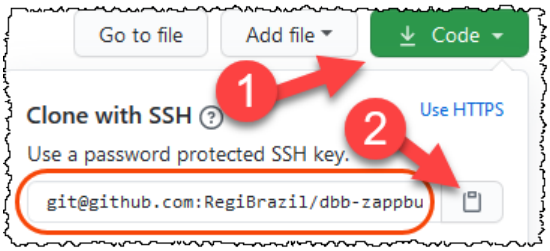


1.2. ⏭ You may "Fork" 

the provided sample from https://github.com/IBM/dbb-zappbuild .
Doing this, I created my repo at https://github.com/RegiBrazil/dbb-zappbuild-1 (this is Private repository that nobody has access).

Click on the green button 

and copy the repo location (must be **SSH)**



1.3. ⏭ Since the repository name will be longer than the provided space, **put the cursor in the field, press PF4,** paste the repo name and press **PF3**:

1.4. ⏭ Paste the USS working directory (where the clone will live). I used **`/var/jenkins/gitispf.`**

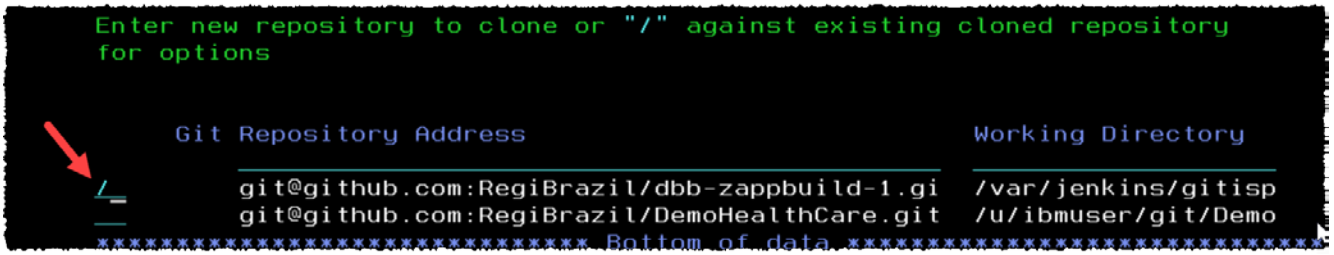
Notice that you also may need to press PF4 if name is longer.

The directory you specify will have an extra directory level added of the repository being cloned. So in the example above, I entered **`/var/jenkins/gitispf`** and the working directory will be **`/var/jenkins/gitispf/dbb-zappbuild-1`**

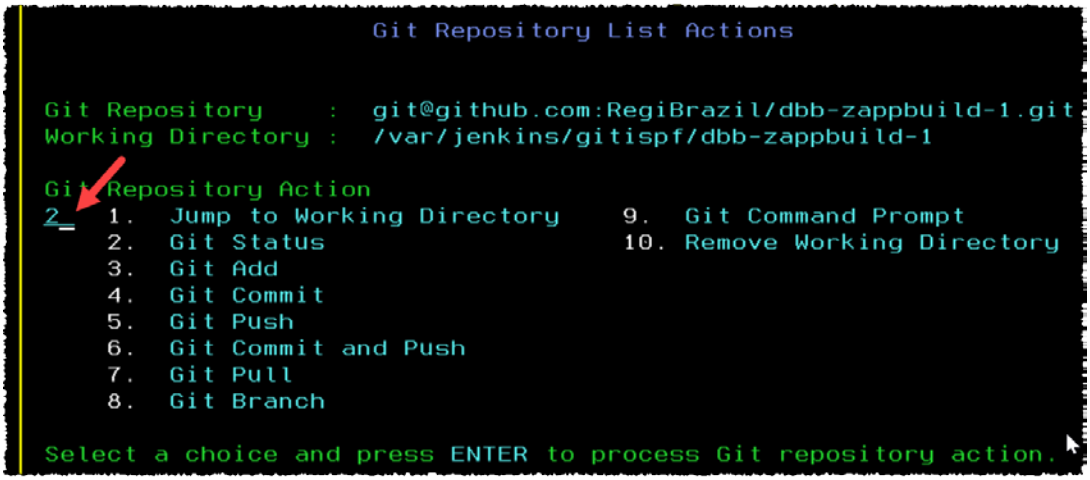
1.5 ⏭ Once you **press enter** the repository will be cloned into the working tree directory.

The folders and files in the repository are cloned into the zFS and not into PDS/PDSE.

1.6. ⏭ Press **PF3** and type **/** at the repository cloned. You may issue Git commands. For example check the status...



1.7. ⏭ Type **2** to see the Git status of this repo:



1.8 Result will be:

2. Working with files in a remote repository

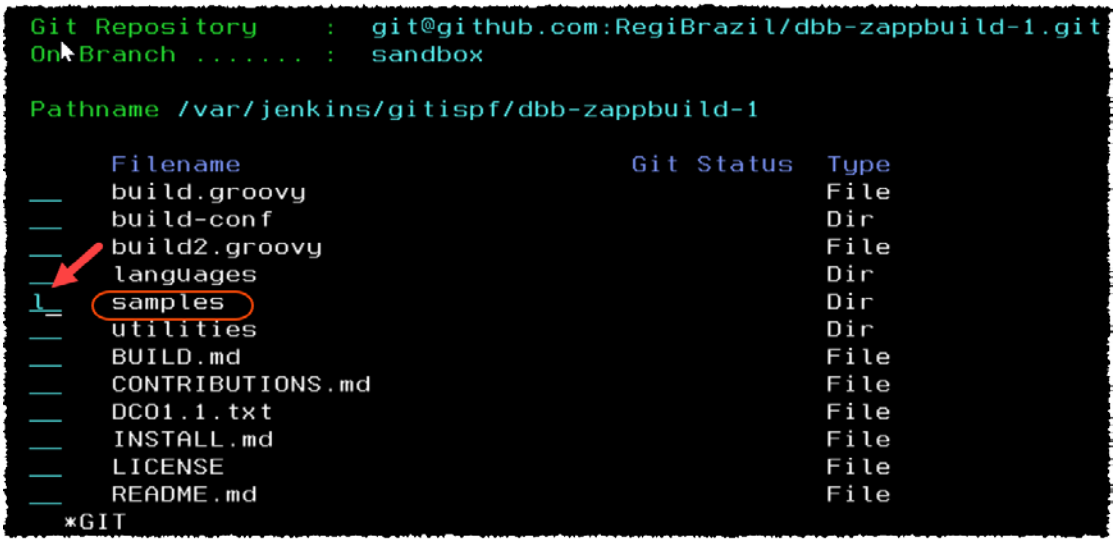
Once a repository is cloned you can choose to work on the files either through the ISPF Git interface or you can work on them directly through ISPF UDLIST (3.17), or using your editor of choice through OMVS..

There are many commands that you may use. See the online doc or use **PF1** for help.

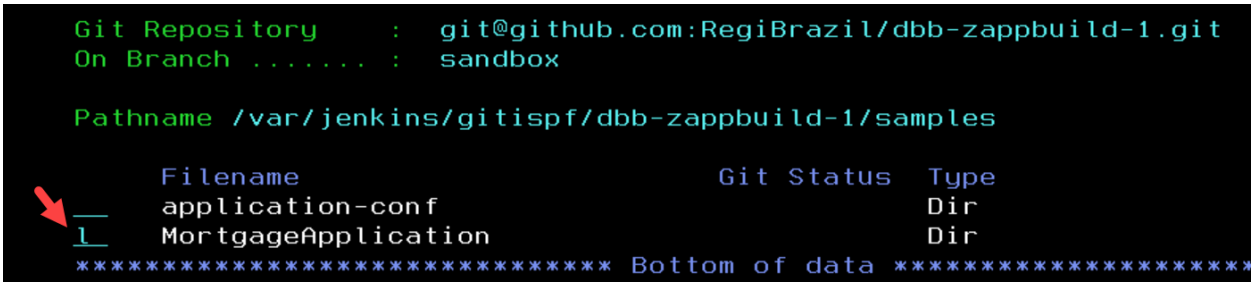
2.1. ⏭ Move the mouse to the field on left of the cloned repo and press **PF1** for the help.
You may use **PF8 and PF7** to advance and return.

2.2. ⏭ To start working with files enter **JU** (Jump to Working Directory) next to the repository that contains your files.

2.3. ⏭ To drill down into a directory enter an **L** next to the directory

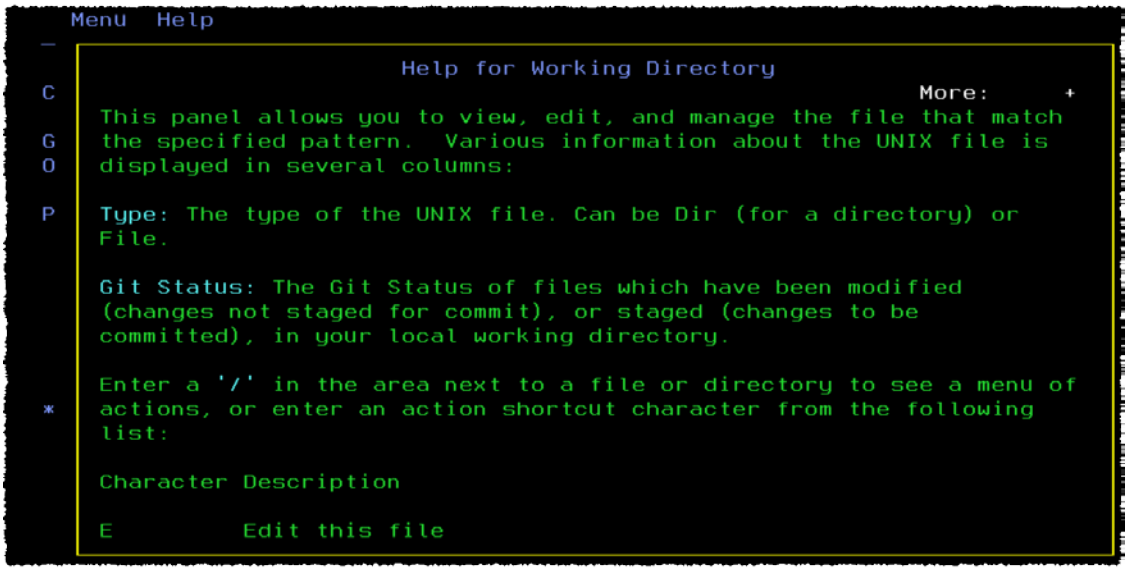


2.4. To drill down into a directory enter an **L** next to the directory



and

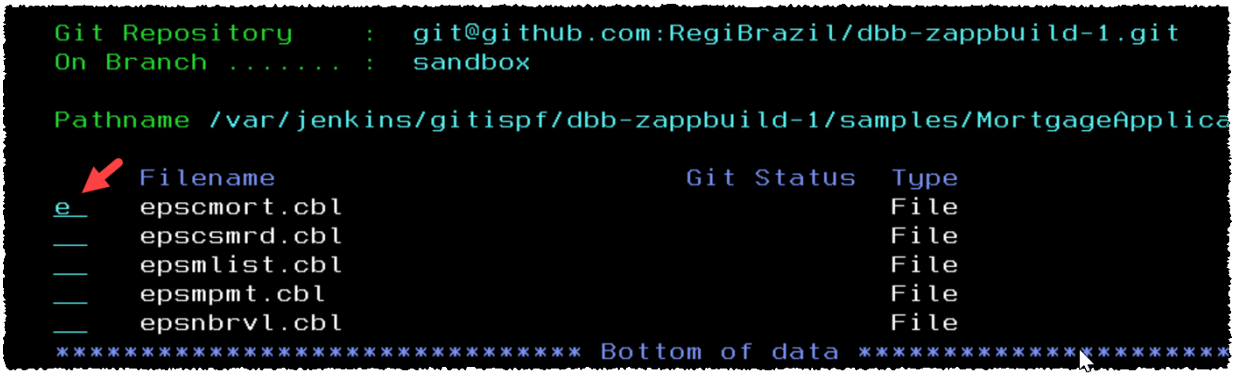
2.5 To see the available options for the file level press **PF1:**



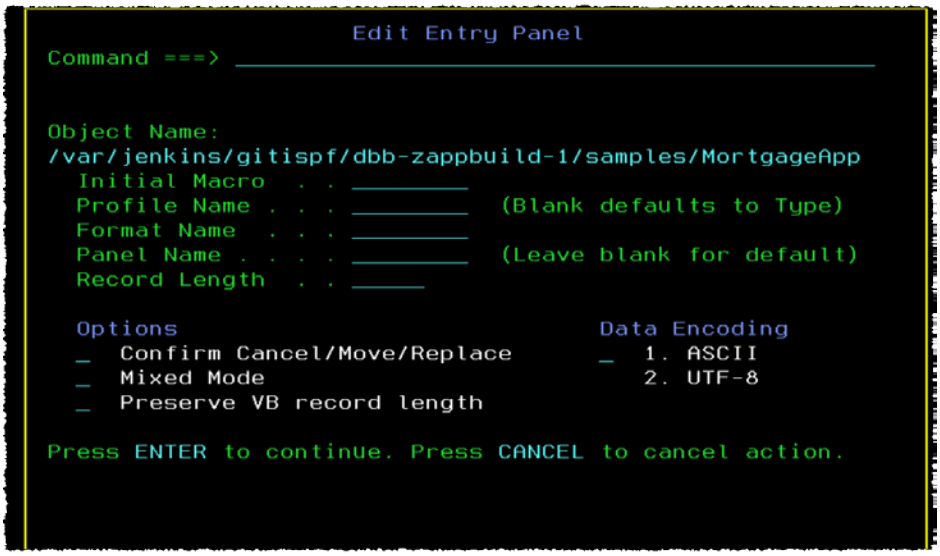
2.6 ⏭ To edit the COBOL using EBCDIC type **E** .

Note: probably would be easier here directly through ISPF UDLIST (3.17), or using your editor of choice through OMVS..

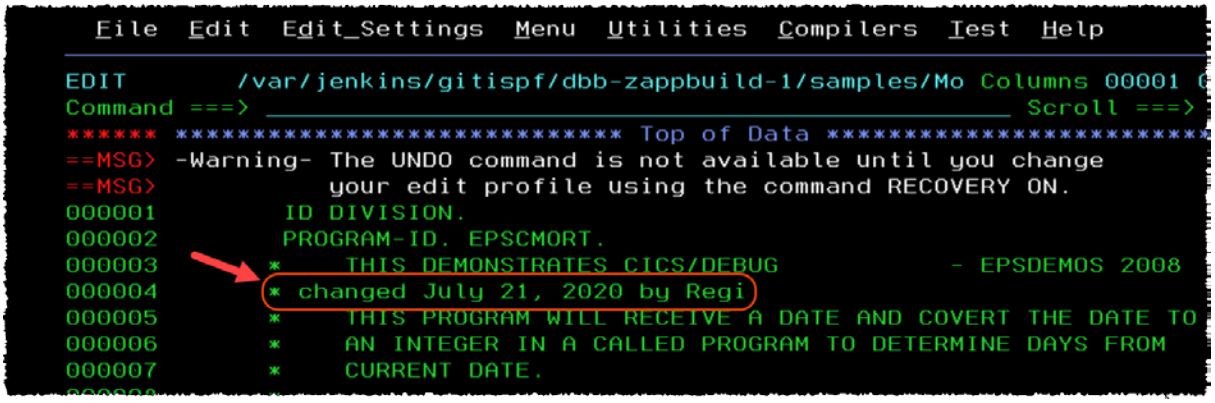
To create a new file enter S on the command line. Make sure you give the filename an extension that matches an entry in your .gitattributes file. This way, when the file is added to staging it will automatically be tagged with the correct CCSID.



⏭ Press **enter**



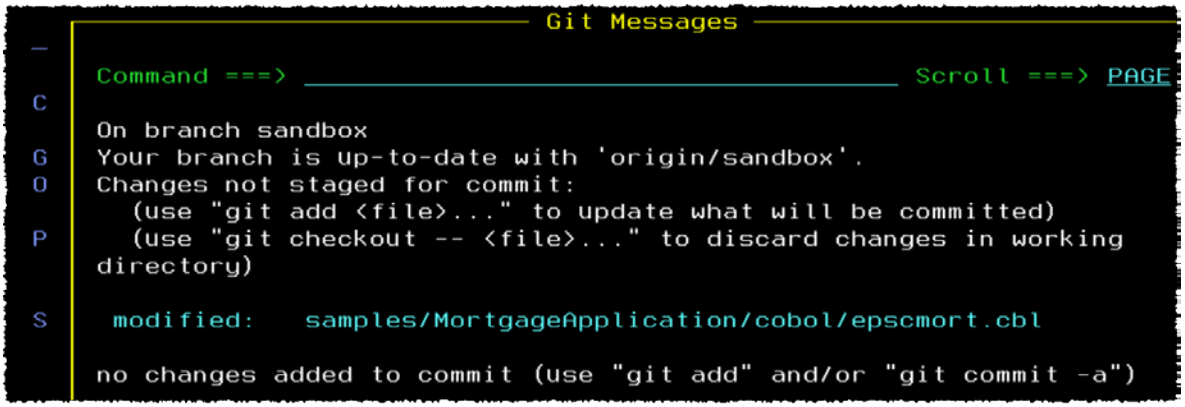
2.7. ⏭ Make a small change in the code as below and press **PF3**



2.8. Notice that the status is changed to "***Modified***"

2.9 ⏭ Type **ST** to see the Git status.

2.10 The result should be as below.
At this point you can try a DBB user build.. You do not need to commit the changes to Git server until you are happy with the results.

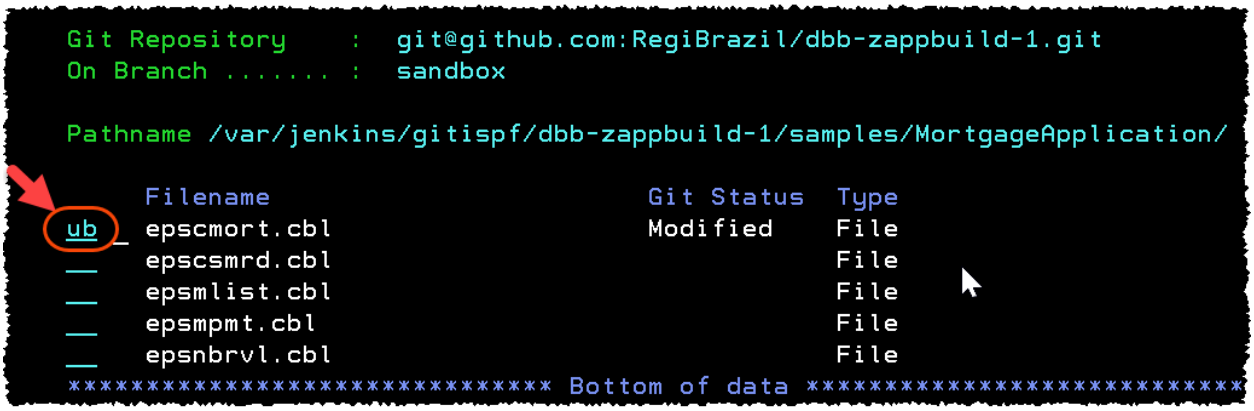


2.11 ⏭ Press **PF3** to dismiss the message

3. Request a DBB User Build through ISPF Git interface

You must enter **UB** (User Build) next to the file you want to build. On the DBB User Build panel, you will have to specify the build script to use (an existing DBB groovy build script that uses *IBM Dependency Based Build*), and the build destination HLQ where you want to have your file built.

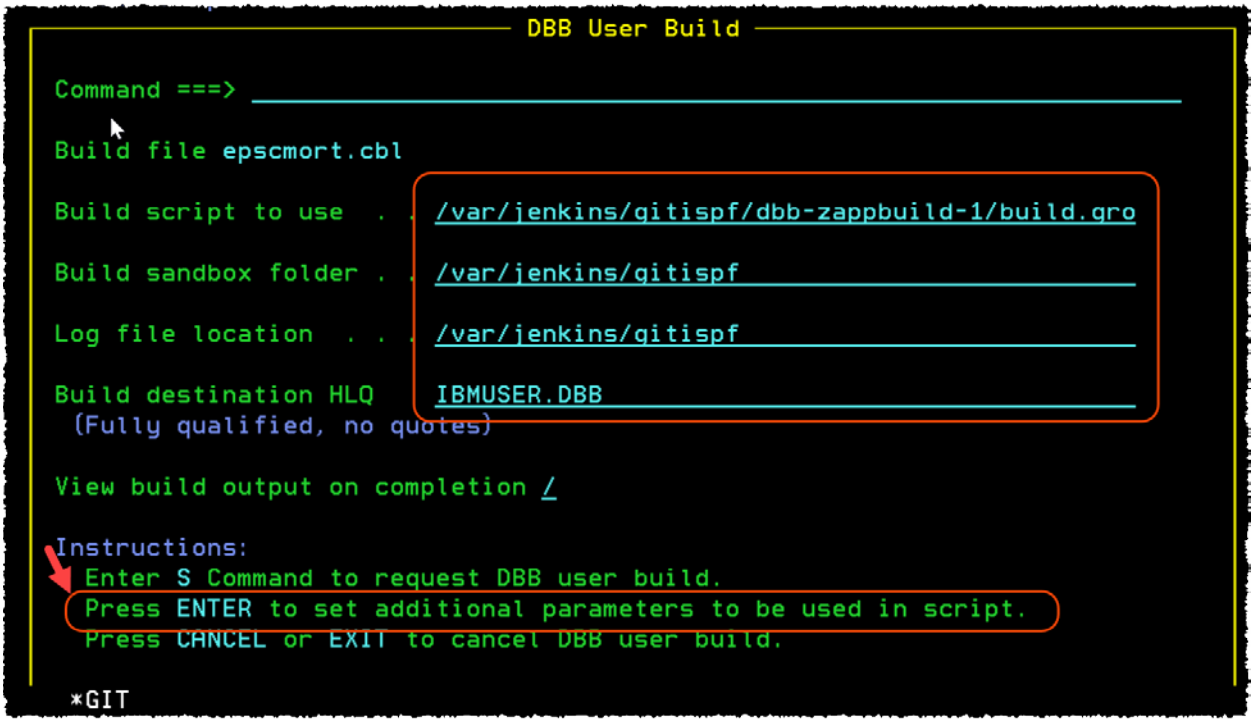
3.1 ⏭ Type **UB** for the COBOL program to be built using *User Build*.



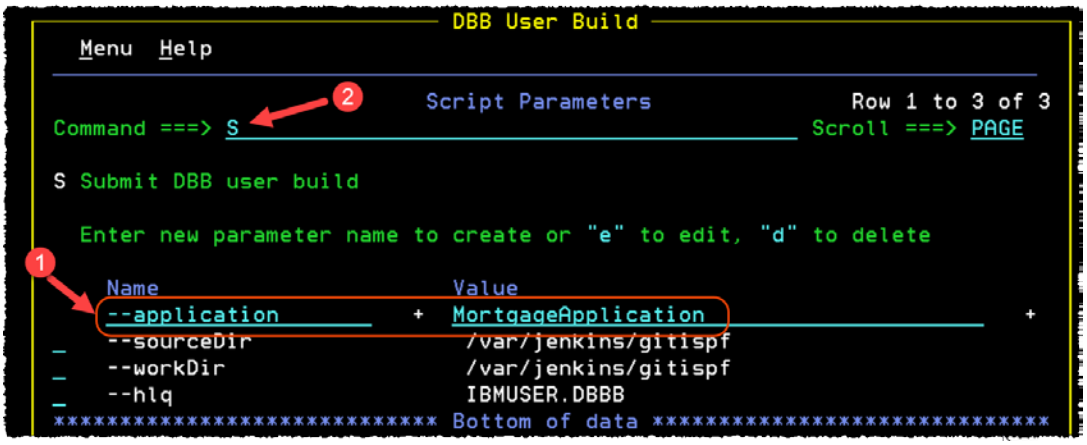
3.2 ⏭ Specify the required data for the User Build

`/var/jenkins/gitispf/dbb-zappbuild-1/build.groovy`
`/var/jenkins/gitispf`
`/var/jenkins/gitispf/work`
`IBMUSER.DBB`

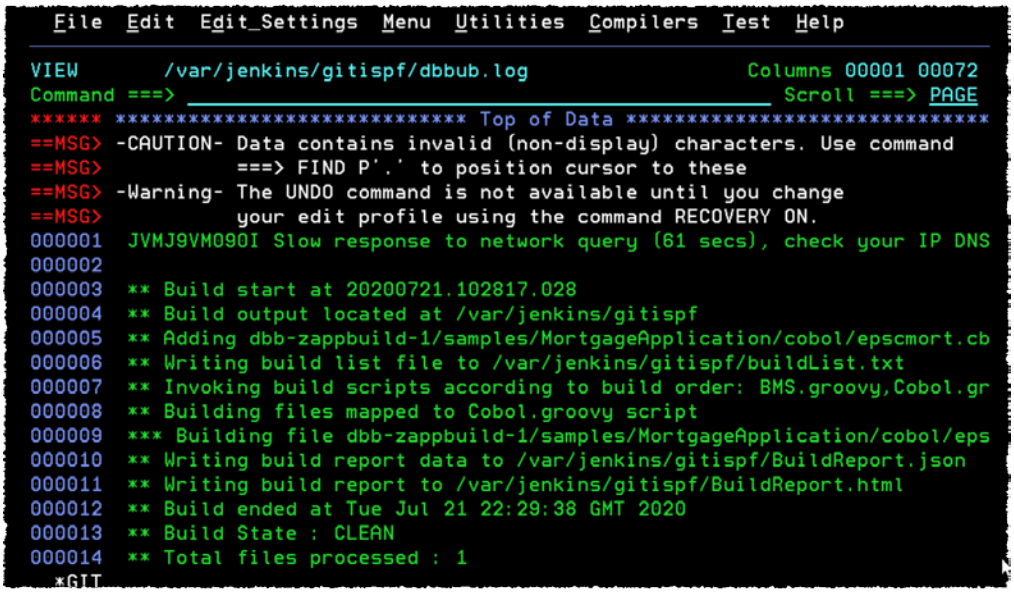
⏭ And press **Enter** to add more parameters..



3.3 Type **–application** parameter and **S** to submit

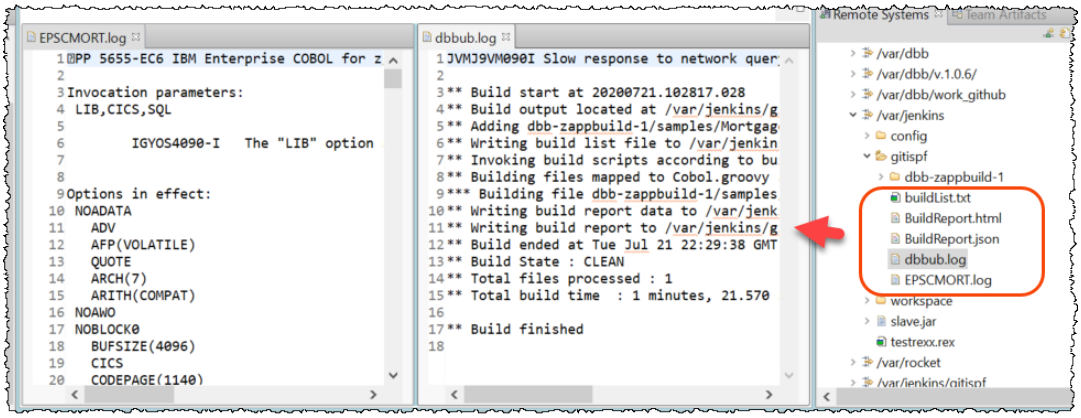


3.4 The *User Build* will start, and the messages will be displayed.

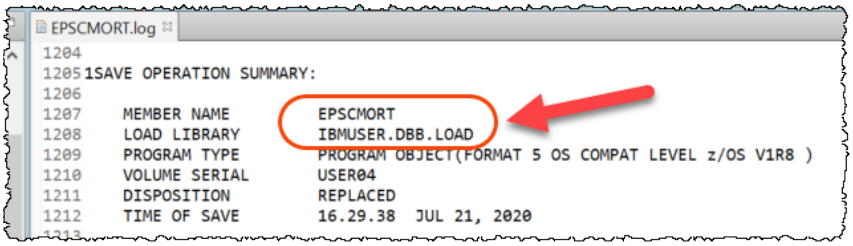


Note : It may have strange non EBCDIC character on the log created. You can ignore that. But if you issue a **`find P'.'`** command you will see that there is a bad character at line 1 (CHARS X'04').

3.5. You may see the log and **buildlist.txt** at the provided uss directory



3.6. ⏭ The z/OS PDS's will be populated:

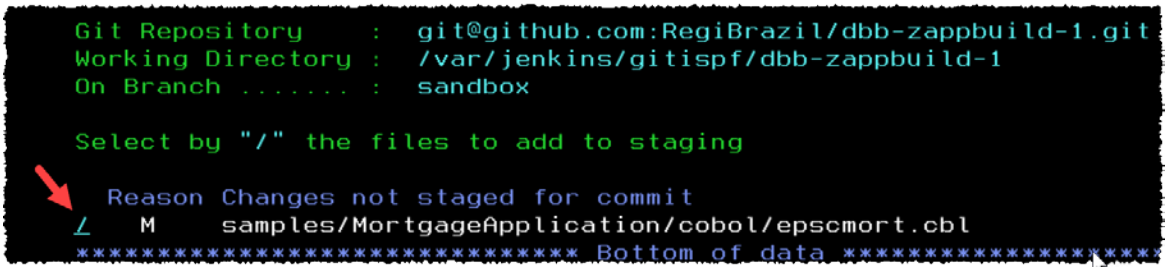


4. Commit and Push the changes to Git Server

Once you are happy wit the results you can **commit** and **push** the changes to the Git Server.

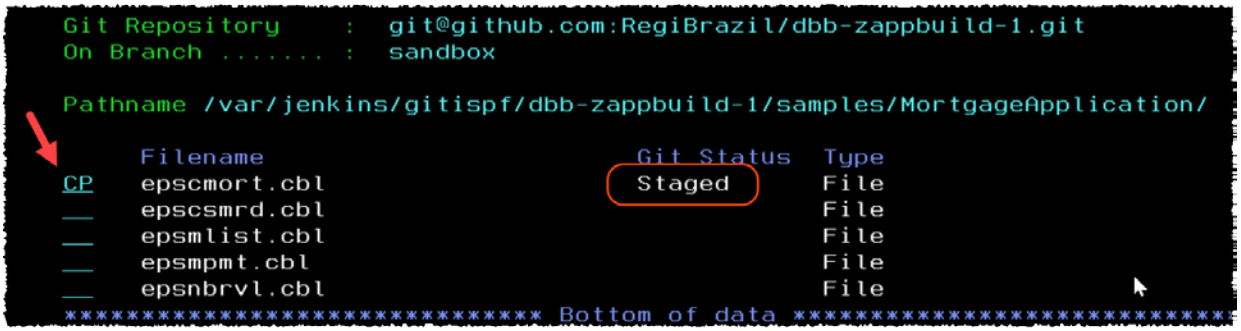
4.1 ⏭ In order to commit the changes you will use the commands:
Git Add – AD (to stage) and **Git Commit and Push – CP**.
This is similar as if you were using IDz.
Type **AD** (Git Add – to stage)

4.2 ⏭ Type **/** to the file to stage

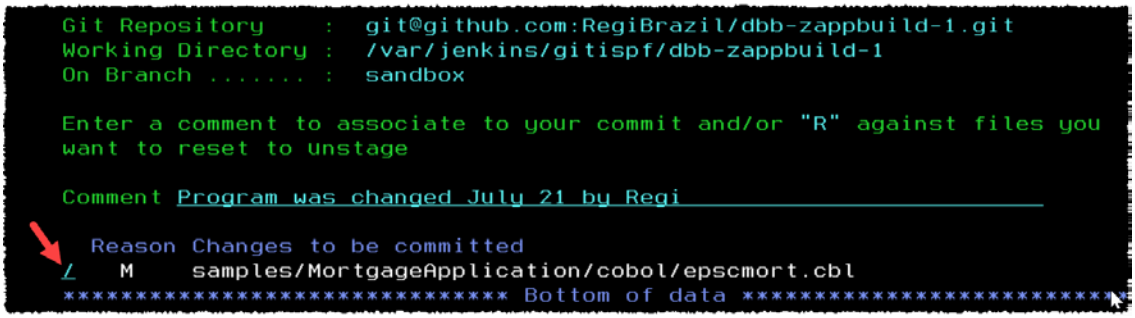


4.3 ⏭ It shows. Press **PF3**

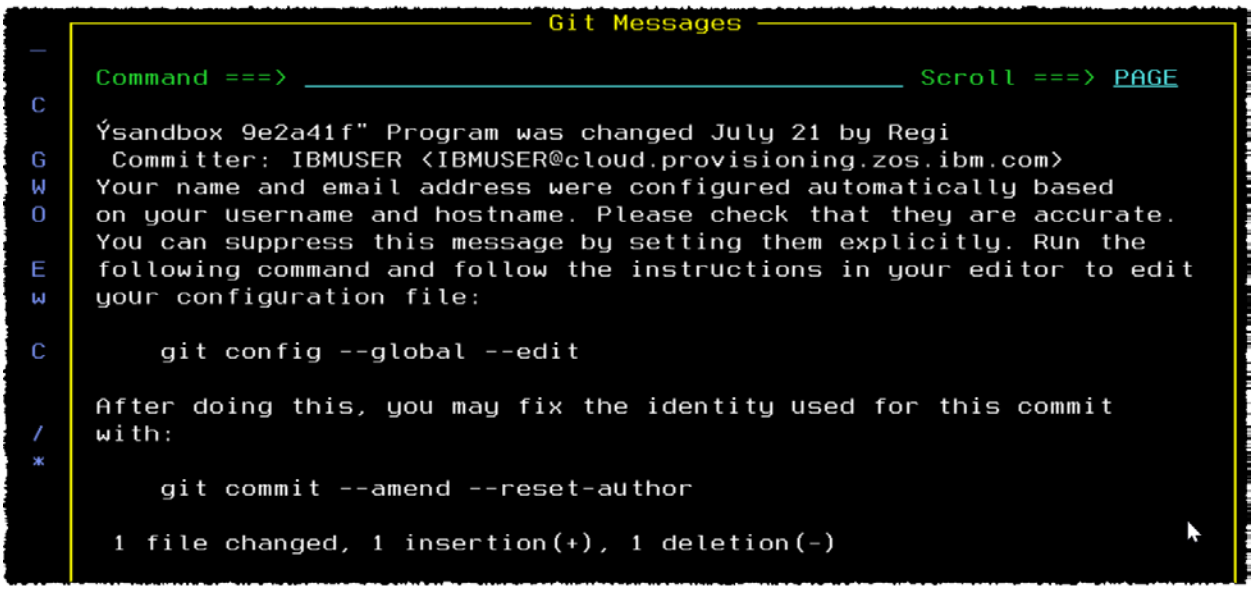
4.4 ⏭ **CP** for *Git Commit and Push.* Notice the Git Status is changed to "***Staged***"



4.5 ⏭ Type **/** to the file to *commit and push*



4.6 ⏭ Press **PF3**



4.7 ⏭ Try now the **ST** (Git Status)

```
Git Repository     :  git@github.com:RegiBrazil/dbb-zappbuild-1.git
On Branch ....... :  sandbox

Pathname /var/jenkins/gitispf/dbb-zappbuild-1/samples/MortgageApplic

    Filename                          Git Status  Type
ST  epscmort.cbl                                  File
__  epscsmrd.cbl                                 File
__  epsmlist.cbl                                 File
__  epsmpmt.cbl                                  File
__  epsnbrvl.cbl                                 File
******************************** Bottom of data ********************
```

```
                          Git Messages
_
   Command ===>
C
   On branch sandbox
G  Your branch is up-to-date with 'origin/sandbox'.
O  nothing to commit, working directory clean
```

4.8 ⏭ If now you go to ***GitHub*** you will see this code comitted/pushed there.