

Managing the code page conversion when migrating z/OS source files to Git

Mathieu Dalbin

mathieu.dalbin@fr.ibm.com

Tim Donnelly

donnell@us.ibm.com

Abstract

This document describes the code page conversion process when migrating source code files from z/OS to Git and provides hints and tips to ensure a successful migration.



Table of content

1	Introduction	3
2	Understanding the code page translation process	4
2.1	Looking inside a file.....	4
2.2	Vocabulary	5
2.3	The challenge of migrating to UTF-8.....	6
2.4	Managing non-roundtripable and non-printable characters.....	6
3	Defining the code page of files in Git.....	9
4	Managing the code page on USS	11
4.1	File tagging.....	11
4.2	Automatic conversion	11
5	Using the DBB Migration Tool	13
5.1	Overview	13
5.2	Set-Up	13
5.3	Migration scenarios	15
5.3.1	Migration using the defaults settings.....	15
5.3.2	Migration using the pdsEncoding Mapping Rule.....	17
5.3.3	Detection of non-roundtripable characters	19
5.3.4	Detection of non-printable characters	21
5.4	Recommendations for using the Migration utility.....	23
6	Conclusion.....	25

1 Introduction

When starting a DevOps modernization journey for the Mainframe platform, the Source Code Management solution chosen for convergence is typically Git using various commercial implementations (GitLab, GitHub, BitBucket, ...). Git has proven to be the de-facto standard in the Open Source world, and the z/OS platform can interact with Git through the z/OS Git client, which is maintained by Rocket Software in its “Open Source Languages and Tools for z/OS” package.

Choosing Git to host enterprise-level software assets implies a migration task of the z/OS artifacts. Whether the source code is under the control of a legacy SCM solution or not, these artifacts are typically stored as members within partitioned data sets (PDSs) with a specific encoding. Depending on locale, country languages, regulations or work habits, the encoding of the source code can differ by various specificities such as accents, locale typography and characters. To support these peculiarities, many different EBCDIC code pages were developed in z/OS and are now widely used.

In the Git world, the standard prevailing encoding is UTF-8, which supports most (if not all) of the characters used in printing worldwide. When migrating artifacts from z/OS to Git, a translation must occur to convert the EBCDIC-encoded content of source files to UTF-8. This process is transparent to the user, whereby the translation is performed by the Rocket Git client, as long as the necessary information on how to perform the conversion is provided. Git uses a specific file called `.gitattributes` to describe and dictate the translation behavior. This file contains details on the encoding that should be used when hosting the artifacts on z/OS or on Git repositories.

This document will highlight some specific migration situations and some cautionary commentary that must be taken into account in order to ensure a successful migration of z/OS artifacts to Git. The purpose of this document is not to cover all the country-related specificities, nor the complex encodings found worldwide. Rather it is aimed at presenting general use cases and how to prevent/circumvent potential pitfalls.

2 Understanding the code page translation process

Code page translation has always been a challenge since z/OS is exchanging data with the distributed systems. To completely understand the effect of encoding, it is important to take a step back and analyze the binary content of files. This analysis will help to better visualize the transformation process that is occurring behind the scenes.

2.1 Looking inside a file...

A file itself can be summarized as being a series of bytes, often expressed in the hexadecimal notation (from x'00' to x'FF'). This file can represent binary content which is not meant to be human-readable (like an executable file), or text content. For the latter and from a user standpoint, this series of bytes doesn't mean anything if the content is not associated with a code page, which will link each byte to a printable character (or a control character). Printable characters are the formation of a character set.

The combination of the contents of a file and the code page used defines the human-readable content. That said, the contents of a file will be displayed slightly differently when combined with different code pages. Let's explore a couple of examples using the ISPF Editor with the "HEX VERT" option enabled to show the hexadecimal values of a file.

Consider the following example where a file containing the French sentence "*Bonjour à tous !*" (which translates to "*Hello everyone!*") is displayed with two different code pages. The file which contains this sentence was saved using IBM-1147 encoding. When the file is then displayed using a different encoding of IBM-1047, although the contents of the file is the same from a binary standpoint, some characters are displayed in a different way:

```
Bonjour à tous !
C9999A9474A9AA44
26516490C036420F
```

Using the IBM-1147 code page

```
Bonjour @ tous |
C9999A9474A9AA44
26516490C036420F
```

Using the IBM-1047 code page

To be correctly displayed with the IBM-1047 code page, the contents of this file must be transformed. Please note how the hexadecimal codes for the 'à' and the '!' characters changed, respectively from x'7C' to x'44' and from x'4F' to x'5A':

```
Bonjour à tous !
C9999A9444A9AA45
265164904036420A
```

Transforming to the IBM-1047 code page

It is important to understand that the code page plays a determining role when displaying the file but also when editing them. To ensure the content of a file is consistent when used by different people, the code page used for editing and displaying will likely be the same for all the users. If Alice edits a file with the IBM-1147 code page and introduces characters (like accents) specific to that code page, then Bob will need to use the IBM-1147 code page to display the content of this file. Otherwise, he may experience the situation depicted earlier, where accents are not displayed correctly. If Bob uses the IBM-1047 code page to edit the file and change the content to replace the special characters by the corresponding accents, then Alice will not be able to display the content of this file like she entered it on her terminal.

Consider this next example. A program file, such as C/370, employs the left ('[') and right (']') brackets in some of the language statement constructs. In many US-based legacy applications, the original encoding used to create source files was IBM-037. If these files are then displayed using a different encoding of IBM-1047, again, although the content of the files is the same from a binary standpoint, the brackets are not displayed the correct way:

```
void main(int argc, char *argv[])
A98849889489A4898864888945898ABB5
569404195D95301973B038190C1975ABD
```

Using the IBM-037 code page

```
void main(int argc, char *argvY")
A98849889489A4898864888945898ABB5
569404195D95301973B038190C1975ABD
```

Using the IBM-1047 code page

To be correctly displayed with the IBM-1047 code page, the contents of this file must be transformed. The hexadecimal codes for the '[' and the ']' characters must be changed from x'BA' and x'BB' to x'AD' and x'BD', respectively:

```
void main(int argc, char *argv[])
A98849889489A4898864888945898AAB5
569404195D95301973B038190C1975DDD
```

Transforming to the IBM-1047 code page

Again, it is very important that anyone and everyone who displays or edits the file uses a consistent code page. This can sometimes be a challenge, as the code page to be used is generally specified in the 3270 Emulator (TN3270) client session set-up. Another challenge is trying to determine the original encoding used to create the file.

To summarize, the binary content of a file must be transformed to ensure consistency when displayed with another code page. This process is known as the code page translation and is key when migrating your source from your z/OS platform to a distributed platform, which is using different code pages (and most likely today, UTF-8).

2.2 Vocabulary

In this document, we will interchangeably use the *coded character set ID* (CCSID), its equivalent name and the *code page* to designate an encoding. Although not strictly equivalent, they are all often interchanged for the sake of convenience. For instance, the IBM-1047 code page is equivalent to the 1047 coded character set (CCSID) and is usually named IBM-1047. The code page for IBM-037 is equivalent to coded character set (CCSID) 037 and is usually named IBM-037. The CCSID for UTF-8 is 1208 and is linked with many code pages.

To simplify, the code pages will be designated by their common name, for instance IBM-037, IBM-1047, IBM-1147 and UTF-8.¹

For reference, a list of code pages is available in the Personal Communications documentation site².

¹ <https://www.ibm.com/docs/en/zos-connect/zosconnect/3.0?topic=properties-coded-character-set-identifiers>

² https://www.ibm.com/docs/en/personal-communications/14.0?topic=SSEQ5Y_14.0.0/com.ibm.pcomm.doc/reference/html/hcp_referencet002.htm

2.3 The challenge of migrating to UTF-8

If you ever tried transferring a z/OS data set to a distributed machine running a traditional operating system (like Microsoft Windows or Linux), you probably ended up with similar content when opening the file:



The content is unreadable because the transfer was performed without a code page conversion. This generally happens when the file was transferred as *binary*, which leaves the content of the file untouched. Most of the transfer utilities offer the capability to transfer the files as binary or as text. In the latter option, code page conversion occurs, and the file should be readable on distributed platforms.

The same conversion must be performed for text files when migrating to Git. Generally, Git uses the UTF-8 code page and, as such, a translation must occur on the content of the transferred files. Likewise, when bringing the files back from Git to z/OS, a backward conversion must be performed to ensure the files are correctly read with the code page in use. Failure in performing this conversion may break the contents of the files, which could become damaged and unreadable. Recovering from this failure is difficult, and the best solution is usually to restart the conversion all over again from the beginning, which often means deleting the files in Git and transferring the same files from z/OS again.

2.4 Managing non-roundtripable and non-printable characters

In the traditional EBCDIC code pages, some characters exist to control the different devices themselves or the way text should be displayed. These characters are located in the first 64 positions of the EBCDIC characters tables³ and can be classified into 2 categories: the *non-printable* characters and the *non-roundtripable* characters. For the former category, these characters can be translated from EBCDIC to UTF-8 and back to EBCDIC without breaking the content of the file. However, editing files which contain these characters with editing tools on a distributed platform, can cause display issues. For the latter category, the translation is still possible from EBCDIC to UTF-8, but the translation back to EBCDIC is not possible. In this case, the translated file is no longer identical to its original version.

³ https://en.wikipedia.org/wiki/EBCDIC#Definitions_of_non-ASCII_EBCDIC_controls

This specific situation must be considered and managed prior to the migration to Git. For these special characters, the conversion from EBCDIC to UTF-8 is generally feasible, but the resulting content of the files may be scrambled or displayed in a wrong way. Moreover, when transferring the files back to z/OS, the files could be damaged, and the transfer could even fail.

In a migration process, when either *non-printable* or *non-roundtripable* characters are found, two options are presented:

- keep these characters “as-is” with the risk of damaging the source files,
- modify the characters to a conversion-friendly format.

For the first option, it is possible to keep the content of the files intact, but the cost of this is to manage the file as a binary artifact in Git. The benefit of this configuration is that the content of the file is not altered by Git when the file is transferred. Because it is flagged as binary, the contents are not converted to UTF-8 and remain the same across the entire workflow (i.e., from z/OS PDS to Git to z/OS PDS). The major drawback of this method is that the modification of these binary files with a distributed platform tool, like an IDE or Notepad, will be extremely difficult. As these files were not converted to UTF-8, their contents will still be in EBCDIC, which is rarely supported outside of the z/OS platform. Browsing or editing the content of these files would require to bring them back to the z/OS platform. Still, this option could be a valid strategy for files that rarely (or never) change, but this implies setting up a strategy when a modification is required (i.e., these files will need to be modified on z/OS, not using a distributed solution - this strategy should remain as an exception).

The alternative is to opt for a transformation of the content that contains *non-printable* or *non-roundtripable* characters before migrating the files. Programmers have often employed elaborate strategies based on the use of these characters in literals or constants. Through the ISPF editor, developers have been able to insert hexadecimal values (when enabling the edition of raw data with the *hex on* feature) in their source code. This use case is very common and can easily be corrected by replacing the hexadecimal coded characters with their corresponding hexadecimal values as shown in the picture below (taken from a CICS Cobol copybook):

```
30      01      DFHBMSCA.
31      02      DFHBMPER PICTURE X VALUE IS '0'.
32      02      DFHBMPNL PICTURE X VALUE IS ' '.
33      02      DFHBMPFF PICTURE X VALUE IS '0'.
34      02      DFHBMPER PICTURE X VALUE IS ' '.
35      02      DFHBMASK PICTURE X VALUE IS '0'.
36      02      DFHBMUNP PICTURE X VALUE IS ' '.
```

Initial values

```
34      01      DFHBMSCA.
35      02      DFHBMPER PICTURE X VALUE IS X'19'.
36      02      DFHBMPNL PICTURE X VALUE IS X'15'.
37      02      DFHBMPFF PICTURE X VALUE IS X'0C'.
38      02      DFHBMPER PICTURE X VALUE IS X'0D'.
39      02      DFHBMASK PICTURE X VALUE IS '0'.
40      02      DFHBMUNP PICTURE X VALUE IS ' '.
```

Final values after transformation

This transformation could even be automated (albeit cautiously) through scripting. Other use cases for these characters should be analyzed carefully, and developers should be enjoined to write their source code in a way that allows for code page conversion.

Fortunately, IBM Dependency Based Build (DBB) provides a feature in its migration script to detect these special characters and helps manage their code page conversion or their transfer as binary.

In any case, the decision about using binary-tagged files in Git, refactoring these files to transform the *non-printable* and *non-roundtripable* characters in their hex values or not changing the content of the files should be taken prior to performing the final migration from datasets to Git repositories. If the migration is started with files that contain either *non-printable* or *non-roundtripable* characters, there is a high risk that files cannot be edited using distributed editors. Once the files are migrated, it is often very difficult to resolve the situation after the fact, as there can be information lost during the code page conversion. In that situation, the best option is to restart the migration from datasets, assuming the original members are still available until the migration is validated.

3 Defining the code page of files in Git

Since Git is a distributed system, each member of this “network” should be able to work with the files stored in the Git repositories. Over the last decade, Git has become the de-facto standard for the development community, being used in all-sized enterprises, internal, open-source and/or personal development projects. Because Git emerged in the distributed world, it was first designed to support technologies from this ecosystem, and it all started with the file encoding. Although it appeared in the 90’s, UTF-8 only gained notoriety in the 2010’s through the increased interconnectivity between heterogeneous systems. UTF-8 is now recognized as the universal, common language for exchanging data and files. Naturally, UTF-8 also became the standard for encoding source files for the distributed world.

Since Mainframe source resides on the host side under the control of a Source Code Management solution, this source is encoded with an EBCDIC code page and could potentially contain national characters. In order to store these source files in Git, a code page conversion must occur between the z/OS source file EBCDIC code page and UTF-8. This is the role of the Rocket Git client on z/OS to perform that conversion, which is transparent for the developers.

However, the Rocket Git client on z/OS must be instructed on the conversion operation to perform. These instructions are defined in a specific file that belongs to the Git repository. The `.gitattributes`⁴ file is a standard file used by Git to declare specific attributes for the Git Repository that it belongs to. With the Rocket Git client on z/OS, it has been extended to describe the code page that should be used for files when they are stored in Git and on z/OS (USS).

The structure of the file is straightforward. One or several attributes are associated to each path and file extension listed in the file. Wildcards can be used for generic definitions. This file must be encoded in ASCII (ISO-8859-1), as it is processed by Git on z/OS and other Git platforms on the distributed side. Here is an example of such a file, which leverages two parameters: `zos-working-tree-encoding` and `git-encoding`.

```
# line endings
* text eol=lf

# file encodings
*.cpy zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.cbl zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.bms zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.pli zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.mfs zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.bnd zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.lnk zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.txt zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.groovy zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.sh zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.properties zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.asm zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.jcl zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.mac zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
*.json zos-working-tree-encoding=utf-8 git-encoding=utf-8
```

The `zos-working-tree-encoding` parameter specifies the code page used for encoding files on USS and in PDSs. It must be consistent with file tags for the files that are under the control of Git. The `git-encoding` parameter specifies the encoding for files stored in Git outside of the z/OS platform. The typical value for this parameter is UTF-8.

⁴ <https://www.git-scm.com/docs/gitattributes>

This file is used by the Rocket Git client to understand which conversion process must be performed when files are added to Git and when files are transferred between a Git repository and the z/OS platform. This file plays an important role in the migration process of source files from PDSs to Git repositories and its content must be thoroughly planned.

An example for this file can be found in the `dbb-zappbuild`⁵ repository. This sample contains the major definitions that are typically found in the context of a migration project.

⁵ <https://github.com/IBM/dbb-zappbuild/blob/main/.gitattributes>

4 Managing the code page on USS

During the migration of source code files from z/OS PDSs to Git, a transitional state exists as the source members are copied to Unix System Services (USS). This step is important and must be correctly performed to ensure the files are imported into Git in the right way.

USS supports the use of different code pages for the files stored on zFS filesystems. As USS introduced the use of ASCII in the z/OS environment, there are some concepts to understand before diving into this section. These concepts are grouped under the term “Enhanced ASCII” in the z/OS documentation⁶.

Files on USS are just a sequence of bytes, like members of PDSs. When members are copied from PDSs to files on USS, the raw content is not altered, and the byte sequence doesn’t change. However, since USS can deal with files coming from distributed systems, which are often encoded in ASCII, additional parameters are introduced to facilitate the management of the file encoding.

4.1 File tagging

One of these parameters is the tagging of files to help determine the nature of the content. By default, files are not tagged on USS, and their content is assumed to be encoded in IBM-1047 by most of the z/OS utilities, unless stated otherwise. If the original content of those files is created with a different code page, some characters may not be rendered or read by programs properly.

A required step to ensure that files are correctly read and processed is to tag them. The tagging of files on USS is controlled by the `chtag` command. With `chtag`, you can print the current tagging information for a file (`-p` option - the `ls -T` command also displays the tag of listed files) or change this tagging (`-t/-c/-b/-m` options). It is important to understand that this tagging information is then leveraged during the migration process to Git by the Rocket Git client, which uses that information to correctly convert the file to the standard Git encoding of UTF-8. Having the correct tagging set for files on USS is a major step in the migration process to ensure a successful conversion.

The following example shows the output of the `ls -aLT` command for a folder that contains Cobol source code:

```
ls -aLT
total 448
          drwxr-xr-x  2 USER1  OMVS      8192 Feb 22 14:47 .
          drwxr-xr-x  7 USER1  OMVS      8192 Jan 12 14:06 ..
t IBM-1047  T=on -rw-r--r--   1 USER1  OMVS      8930 Feb 22 13:04 epscmort.cbl
t IBM-1047  T=on -rw-r--r--   1 USER1  OMVS    132337 Jan 12 14:06 epscmrd.cbl
t IBM-1047  T=on -rw-r--r--   1 USER1  OMVS      7919 Feb 22 13:04 epsmlist.cbl
t IBM-1047  T=on -rw-r--r--   1 USER1  OMVS      5854 Jan 12 14:06 epsmpmt.cbl
t IBM-1047  T=on -rw-r--r--   1 USER1  OMVS      6882 Jan 12 14:06 epsnbrvl.cbl
```

4.2 Automatic conversion

Although not directly involved in the migration process, the Enhanced ASCII feature introduces an in-flight automatic conversion for files stored in USS. This automatic conversion is mainly controlled by the `_BPXK_AUTOCVT` environment variable or by the `AUTOCVT` parameter defined in a `BPXPRMxx` PARMLIB member. By default, programs on USS are operating in EBCDIC. When the `_BPXK_AUTOCVT` parameter is activated to either ON or ALL, along with the right tagging of files, programs executing in USS can transparently and seamlessly work with ASCII files without converting them to EBCDIC.

⁶ <https://www.ibm.com/docs/en/zos/2.5.0?topic=pages-enhanced-ascii>

During the migration process to Git, a special file called `.gitattributes` is used to describe the conversion format for all the files under the control of Git. This file must be encoded in ASCII (ISO-8859-1), as it is processed by Git on z/OS and other Git platforms on the distributed side. To avoid any manual tasks, it is recommended to enable the automatic conversion for any thread working with Git on z/OS and the `.gitattributes` file. This file will be discussed in further detail in section “[3 Defining the code page of files in Git](#)”.

Other options can interfere in the automatic conversion process. Without being exhaustive, this list provides some parameters which can also influence the global behavior of tools ported to z/OS, including the Rocket Git client for z/OS:

<code>_TAG_REDIRECT_IN</code> <code>_TAG_REDIRECT_OUT</code> <code>_TAG_REDIRECT_ERR</code>	https://www-40.ibm.com/servers/resourcelink/svc00100.nsf/pages/zosV2R4SA232280/\$file/bpxa500_v2r4.pdf
<code>FILETAG</code>	https://www.ibm.com/docs/en/zos/2.3.0?topic=options-filetag-cc-only

These parameters can affect the overall behavior of the USS environment, so they should be configured with caution. Detailing the impacts of these parameters is outside the scope of this document. Refer to the above documentation to understand how they can affect your configuration. Additionally, the recommended values for these parameters are described in the DBB configuration documentation page⁷.

⁷ <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=dbb-setting-up-git-uss>

5 Using the DBB Migration Tool

IBM® Dependency Based Build (DBB) provides a migration tool to help facilitate the copying of source artifacts to Git from your z/OS development SCM. This tool can be used to copy the source code contained in your z/OS partitioned data sets (PDSs) to a local Git repository on Unix System Services (USS) that will later be committed and pushed to a distributed Git server.

The migration tool is bundled as part of the SMP/e installation of the Dependency Based Build toolkit (FMID HBGZ110) and is typically found under the `migration/bin` sub-directory⁸ of the DBB installation folder (which by default is `/usr/lpp/IBM/dbb`, unless customized during installation). The setup along with various options provided by this tool to assist in the migration can be found on the IBM Documentation website⁹.

5.1 Overview

The primary purpose of the migration tool is to copy the source code PDS members into a pre-existing local Git repository in USS, stored on the z/OS File System (zFS). During this copy, the tool will perform the necessary code page conversion from the z/OS code page to the traditional code page used in Git when applicable, create (or update) the `.gitattributes` file with the correct encoding mappings, tag the USS files with the appropriate code page values and (potentially) report on any issues encountered during the migration.

The following sections will showcase various migration scenarios using the context that has been described in the previous sections. The intent is not to provide an exhaustive set of scenarios supported by the migration tool, but rather to focus on common use cases. The scenarios are:

1. Migration using the defaults settings
2. Migration using the `pdsEncoding` Mapping Rule
3. Detection of `non-roundtripable` characters
4. Detection of `non-printable` characters

5.2 Set-Up

To illustrate the above scenarios, the following PDSs were constructed to highlight some specific migration situations that may be encountered and how to mitigate potential issues.

Content of the `MIGRATE.TECHDOC.SOURCE` dataset:

Member	Description
IBM037	<p>Member that has been created using the code page of IBM-037.</p> <p>Example: IBM-037 Code Page <code>void main(int argc, char *argv[])</code></p> <p>Note: Under the IBM-037 code page, the hexadecimal codes for the <code>'I'</code> and the <code>'J'</code> characters are <code>x'BA'</code> and <code>x'BB'</code>; respectively.</p>
IBM1047	<p>Member that has been created using the code page of IBM-1047.</p> <p>Example: IBM-1047 Code Page <code>void main(int argc, char *argv[])</code></p> <p>Note: Under the IBM-1047 code page, the hexadecimal codes for the <code>'I'</code> and the <code>'J'</code> characters are <code>x'AD'</code> and <code>x'BD'</code>; respectively.</p>

⁸ <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=dbb-installing-configuring-toolkit-zos>

⁹ <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git>

Content of the *MIGRATE.TECHDOC.COPYBOOK* dataset:

Member	Description
NROUND	<p>Member that contains non-roundtripable characters.</p> <p>Example: Non-Roundtripable Characters Line with CHAR_NL (0x15) ☒ Line with CHAR_CR (0x0D) ☒ Line with CHAR_LF (0x25) ☒ Line with CHAR_SHIFT_IN (0x0F) _ Line with CHAR_SHIFT_OUT(0x0E) _ Line with empty CHAR_SHIFT_OUT(0x0E) and CHAR_SHIFT_IN (0x0F) _</p>
NPRINT	<p>Member that contains non-printable characters.</p> <p>Example: Non-Printable Characters Line with (0x06) ? Line with (0x07) ☒ Line with (0x1B) ☒</p>
HEXCODED	<p>Member that contains non-printable and non-roundtripable characters.</p> <p>Example: Hexadecimal Coded Characters 01 DFHBMSCA. 02 DFHBMPPEM PICTURE X VALUE IS ' _'. 02 DFHBMPNL PICTURE X VALUE IS '☒'. 02 DFHBMPFF PICTURE X VALUE IS ' _'. 02 DFHBMPCR PICTURE X VALUE IS '☒'. 02 DFHBMASK PICTURE X VALUE IS '0'. 02 DFHBMUNP PICTURE X VALUE IS ' '.</p>
HEXVALUE	<p>Member that contains the suggested transformation of the non-printable or non-roundtripable characters contained in the "HEXCoded" member in a more suitable format.</p> <p>Example: Hexadecimal Values 01 DFHBMSCA. 02 DFHBMPPEM PICTURE X VALUE IS X'19'. 02 DFHBMPNL PICTURE X VALUE IS X'15'. 02 DFHBMPFF PICTURE X VALUE IS X'0C'. 02 DFHBMPCR PICTURE X VALUE IS X'0D'. 02 DFHBMASK PICTURE X VALUE IS '0'. 02 DFHBMUNP PICTURE X VALUE IS ' '.</p>

5.3 Migration scenarios

5.3.1 Migration using the defaults settings

In this scenario we will be migrating all the source members in `MIGRATE.TECHDOC.SOURCE` PDS using the default settings into a local USS Git Repository under `/u/user1/Migration`. It is the most simplistic form of invoking the migration tool and, in most cases, satisfies most needs.

```
$DBB_HOME/migration/bin/migrate.sh -r /u/user1/Migration -m Mapping
Rule[h1q:MIGRATE.TECHDOC,extension:SRC,toLower:true] SOURCE

Setting dbb.file.tagging = true
Local GIT repository: /u/user1/Migration
Mapping: MappingRule[h1q:MIGRATE.TECHDOC,extension:SRC,toLower:true]
MappingRuleId: com.ibm.dbb.migration.MappingRule
MappingRuleAttrs: [h1q:MIGRATE.TECHDOC, extension:SRC, toLower:true]
Using mapping rule com.ibm.dbb.migration.MappingRule to migrate the data sets
Migrating data set SOURCE
Copying MIGRATE.TECHDOC.SOURCE(IBM037) to /u/user1/Migration/source/ibm037.src using default
encoding
Copying MIGRATE.TECHDOC.SOURCE(IBM1047) to /u/user1/Migration/source/ibm1047.src using default
encoding
** Build finished
```

Note that the migration tool is using a default encoding, which is IBM-1047.

An examination of the files on the local USS Git repository will reveal that the files were copied and were tagged with the default code page of IBM-1047.

```
ls -alT /u/user1/Migration/source
total 64
          drwxr-xr-x  2 USER1  OMVS      8192 May  4 12:33 .
          drwxr-xr-x  4 USER1  OMVS      8192 May  4 12:33 ..
t IBM-1047  T=on  -rw-r--r--  1 USER1  OMVS         61 May  4 12:33 ibm037.src
t IBM-1047  T=on  -rw-r--r--  1 USER1  OMVS         61 May  4 12:33 ibm1047.src
```

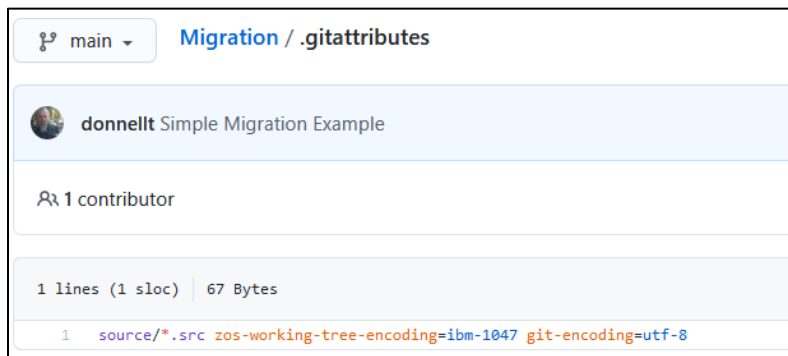
Additionally, the `.gitattributes` file was created (or updated) with the correct encoding mappings. All source artifacts, except those tagged as binary (to be discussed later), will be stowed in the distributed Git server using the UTF-8 code page (as defined by the `git-encoding=utf-8` parameter), whereas any artifacts that are copied from the distributed Git server to z/OS will be translated to the IBM-1047 code page (as defined by the `zos-working-tree-encoding=ibm-1047` parameter). The Section “[3 Defining the code page of files in Git](#)” provides more information.

```
cat /u/user1/Migration/.gitattributes
source/*.src zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
```

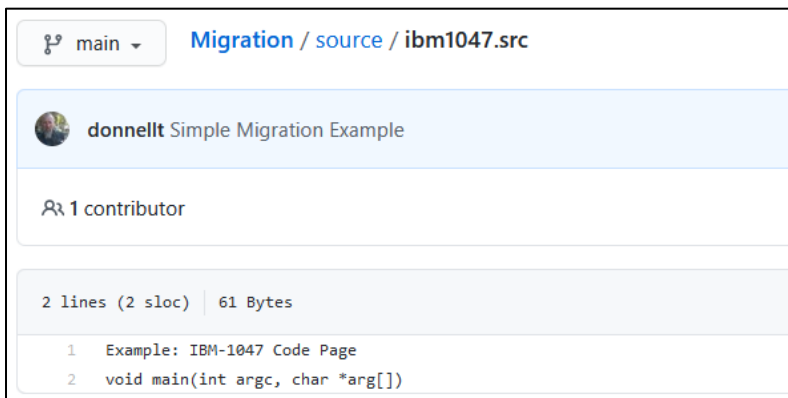
At this point, Git actions like add/commit/push can be performed on the migrated source artifacts to the distributed Git server.

```
git add .
git commit -m "Simple Migration Example"
[main 6436b92] Simple Migration Example
 3 files changed, 5 insertions(+)
 create mode 100644 .gitattributes
 create mode 100644 source/ibm037.src
 create mode 100644 source/ibm1047.src
git push
Counting objects: 6, done.
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 634 bytes | 211.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
To github.ibm.com:user1/Migration.git
 3d2962a..6436b92 main -> main
```

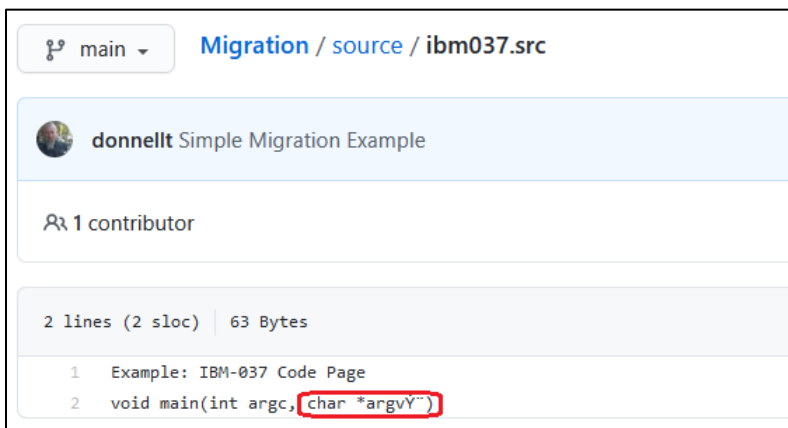
Once the Git push command has completed to the distributed Git server, the resulting files should be translated into the correct UTF-8 code page.



The screenshot shows the GitHub interface for the file `Migration/.gitattributes`. The repository is named "Migration" and the file is located at the root. The commit is by "donnellt" with the message "Simple Migration Example". There is 1 contributor. The file has 1 line (1 sloc) and is 67 Bytes. The content of the file is: `1 source/*.src zos-working-tree-encoding=ibm-1047 git-encoding=utf-8`



The screenshot shows the GitHub interface for the file `Migration/source/ibm1047.src`. The repository is named "Migration" and the file is located at `source/ibm1047.src`. The commit is by "donnellt" with the message "Simple Migration Example". There is 1 contributor. The file has 2 lines (2 sloc) and is 61 Bytes. The content of the file is: `1 Example: IBM-1047 Code Page
2 void main(int argc, char *arg[])`



The screenshot shows the GitHub interface for the file `Migration/source/ibm037.src`. The repository is named "Migration" and the file is located at `source/ibm037.src`. The commit is by "donnellt" with the message "Simple Migration Example". There is 1 contributor. The file has 2 lines (2 sloc) and is 63 Bytes. The content of the file is: `1 Example: IBM-037 Code Page
2 void main(int argc, char *argv")` The `char *argv"` part of the second line is highlighted with a red box.

However, as indicated in the last picture above , the `ibm037.src` file reveals an encoding issue. This will be discussed in the next scenario.

5.3.2 Migration using the `pdsEncoding` Mapping Rule

In this scenario, we will be migrating a single source member from the `MIGRATE.TECHDOC.SOURCE` PDS using the `pdsEncoding` keyword in the mapping rule to override the default encoding. Recall that from the previous migration scenario, there was an encoding issue with the final copy of the `ibm037.src` file in the distributed Git server. This occurred because the z/OS file was written using the IBM-037 code page instead of the default IBM-1047 code page. The problem is not in how the file was encoded, but rather how the Rocket Git client converted the file when sending it to Git.

This can be a common occurrence for source files that pre-date the introduction of Unix System Services (USS), and where high-level languages, such as C/370, were utilized. As stated previously, determining the original encoding can be a challenge since the code page used to create the file is generally specified in the 3270 Emulator (TN3270) client session set-up. Therefore, an analysis of the z/OS source should be performed to determine the original code page used to create the source. To determine the code page used to create files through ISPF, an alternate option is to ask the developers which code page they are using to edit the files through their 3270 connections.

To correct the encoding issue identified in the previous scenario, we will use the `pdsEncoding` keyword of the mapping rule to override the default IBM-1047 code page with IBM-037 for the offending member.

```
$DBB_HOME/migration/bin/migrate.sh -r /u/user1/Migration -m
MappingRule[hlq:MIGRATE.TECHDOC,extension:SRC,toLower:true,pdsEncoding:IBM-037]
"SOURCE(IBM037)"

Setting dbb.file.tagging = true
Local GIT repository: /u/user1/Migration
Mapping: MappingRule[hlq:MIGRATE.TECHDOC,extension:SRC,toLower:true,pdsEncoding:IBM-037]
MappingRuleId: com.ibm.dbb.migration.MappingRule
MappingRuleAttrs: [hlq:MIGRATE.TECHDOC, extension:SRC, toLower:true, pdsEncoding:IBM-037]
Using mapping rule com.ibm.dbb.migration.MappingRule to migrate the data sets
Migrating data set SOURCE(IBM037)
Copying MIGRATE.TECHDOC.SOURCE(IBM037) to /u/user1/Migration/source/ibm037.src using IBM-037
** Build finished
```

Note that the migration tool is using the override encoding of IBM-037 for a named member. This override does not necessarily have to be performed on a member-by-member basis, as the migration tool supports the ability to override the encoding for an entire PDS being migrated.

An examination of the files on the local USS Git repository will reveal that the file was copied and tagged with the override code page of IBM-037.

```
ls -alT /u/user1/Migration/source
total 64
          drwxr-xr-x  2 USER1  OMVS      8192 May  4 13:10 .
          drwxr-xr-x  4 USER1  OMVS      8192 May  4 13:10 ..
t IBM-037  T=on  -rw-r--r--  1 USER1  OMVS         61 May  4 14:54 ibm037.src
t IBM-1047 T=on  -rw-r--r--  1 USER1  OMVS         61 May  4 13:10 ibm1047.src
```

Additionally, the `.gitattributes` file was updated with the correct encoding mappings.

```
cat /u/user1/Migration/.gitattributes
source/*.src zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
source/*.src zos-working-tree-encoding=IBM-037 git-encoding=utf-8
```

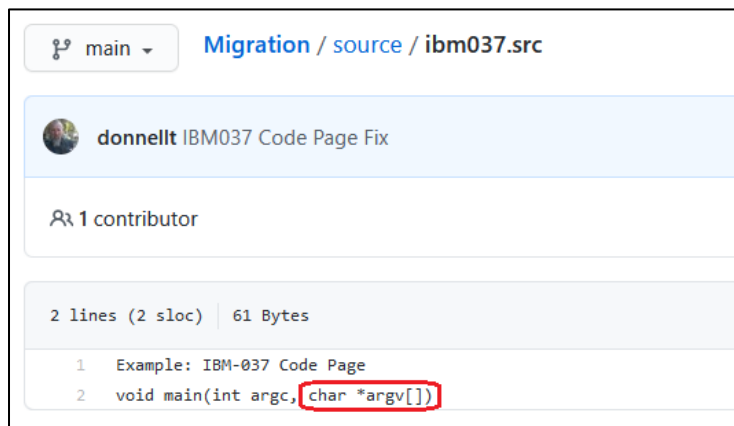
However, in this example you will notice a slight anomaly in that there are two (2) entries for the same sub-folder `source/*.src`. This will cause an encoding conflict during the Git add action. To correct this situation, the `.gitattributes` file must be manually updated to add the file name. Wild cards can be use in the file name should there be more than one member that matches this situation. The order of these entries is important, with the last entry taking precedence. In some cases, additional wild carding may be required to prevent further conflicts.

```
cat /u/user1/Migration/.gitattributes
source/*.src zos-working-tree-encoding=ibm-1047 git-encoding=utf-8
source/ibm037.src zos-working-tree-encoding=IBM-037 git-encoding=utf-8
```

Once the correction has been made to the `.gitattributes` file, the Git commit and push actions can be performed on the updated files to the distributed Git server.

```
git add .
git commit -m "IBM037 Code Page Fix"
[main 107c86c] IBM037 Code Page Fix
 2 files changed, 2 insertions(+), 1 deletion(-)
git push
Counting objects: 5, done.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 485 bytes | 485.00 KiB/s, done.
Total 5 (delta 2), reused 0 (delta 0)
To github.ibm.com:user1/Migration.git
 6436b92..107c86c main -> main  3d2962a..6436b92 main -> main
```

Now when examining the offending file on the distributed Git server, the contents of the file should be translated correctly.



The probability that members of a single PDS were written using a different code page, though possible, is extremely low. However, it is worth pointing out that it could expose an issue in how the migration tool generates the `.gitattributes` file.

5.3.3 Detection of non-roundtripable characters

In this scenario, we will examine how the migration tool can assist in the detection of what is known as *non-roundtripable* characters. The section “[2.4 Managing non-roundtripable and non-printable characters](#)” provides more information. To illustrate this, we will be migrating the single source member `MIGRATE.TECHDOC.COPYBOOK(NROUND)`, which contains both types of characters.

During the migration of a PDS member to a USS file, the migration tool will scan the content of the file to see if it detects any *non-roundtripable* characters. These characters are defined in the `migrate.groovy` script and are:

```
@Field def CHAR_NL = 0x15
@Field def CHAR_CR = 0x0D
@Field def CHAR_LF = 0x25
@Field def CHAR_SHIFT_IN = 0x0F
@Field def CHAR_SHIFT_OUT = 0x0E
```

If detected, the migration tool will emit a diagnostic message in the console log and will copy the member to USS as *binary* and therefore no code page conversion will be performed.

```
$DBB_HOME/migration/bin/migrate.sh -r /u/user1/Migration -m
MappingRule[h1q:MIGRATE.TECHDOC,extension:CPY,toLower:true,pdsEncoding:IBM-037]
"COPYBOOK(NROUND)"

Local GIT repository: /u/user1/Migration
Using mapping rule com.ibm.dbb.migration.MappingRule to migrate the data sets
Migrating data set COPYBOOK(NROUND)
[WARNING] Copying MIGRATE.TECHDOC.COPYBOOK(NROUND) to /u/user1/Migration/copybook/nround.cpy
! Possible migration issue:
  Line 2 contains non-roundtripable characters:
    Char 0x15 at column 27
  Line 3 contains non-roundtripable characters:
    Char 0x0D at column 27
  Line 4 contains non-roundtripable characters:
    Char 0x25 at column 27
  Line 7 contains non-roundtripable characters:
    Empty Shift Out and Shift In at column 74

! Copying using BINARY mode
** Build finished
```

Note that the migration tool has detected numerous *non-roundtripable* characters on various lines and has performed the copy as binary.

An examination of the files on the local USS Git Repository will reveal that the file was copied but left untagged (this is a current known limitation for the DBB Toolkit in its 1.1.3 version, and a workaround is available upon request).

```
ls -alT /u/user1/Migration/copybook
total 48
          drwxr-xr-x  2 USER1  OMVS      8192 May  6 13:42 .
          drwxr-xr-x  4 USER1  OMVS      8192 May  6 13:42 ..
- untagged  T=off -rw-r--r--  1 USER1  OMVS       560 May  6 13:42 nround.cpy
```

To be processed by the Rocket Git client when performing the Git add command, the file should be manually tagged as binary first. To correctly tag the file as `binary`, use the `chtag -b` command prior to performing the Git add command.

```
ls -alT /u/user1/Migration/copybook
total 48
      drwxr-xr-x  2 USER1  OMVS      8192 May  8 13:10 .
      drwxr-xr-x  5 USER1  OMVS      8192 May  8 13:12 ..
b binary      T=off -rw-r--r--  1 USER1  OMVS       560 May  8 13:10 nround.cpy
```

Additionally, the `.gitattributes` file was automatically updated by the migration script to indicate that the file is mapped as `binary`.

```
cat /u/user1/Migration/.gitattributes
copybook/nround.cpy binary
```

During the Git push to the distributed Git server, Git will treat this as a binary file and no conversion to UTF-8 will take place. In essence, the resulting file on the distributed Git server will be the original contents of the PDS member, in EBCDIC.

```
git add .
warning: copybook/nround.cpy added file have been automatically tagged BINARY because they
were untagged yet the .gitattributes file specifies they should be tagged
git commit -m "Binary File"
[main 0213795] Binary File
 2 files changed, 1 insertion(+)
 create mode 100644 .gitattributes
 create mode 100644 copybook/nround.cpy
git push
Counting objects: 5, done.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 598 bytes | 598.00 KiB/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To github.ibm.com:user1/Migration.git
 e90192..0213795 main -> main
```

Once the Git push action has completed to the distributed Git server, the resulting file will be treated as binary.



This may not be an ideal situation as described in the section “2.4 Managing non-roundtripable and non-printable characters” and should be corrected/reconciled before continuing with the migration.

5.3.4 Detection of non-printable characters

In this final scenario, we will examine how the migration tool can assist in the detection of what is known as *non-printable* characters. The section “[2.4 Managing non-roundtripable and non-printable characters](#)” provides more information. To illustrate this, we will again migrate the single source member `MIGRATE.TECHDOC.COPYBOOK(NPRINT)`, which only contains *non-printable* characters.

During the migration of a PDS member to a USS file, the migration tool will scan the content of the file to see if it detects any *non-printable* characters. These characters are defined as any hexadecimal values that are an EBCDIC x'40' or less, and not one of the five (5) *non-roundtripable* characters.

The migration tool provides three (3) options on how to handle and report on these characters.

1. Do Not Check - The *non-printable* characters are not researched; the member is simply copied as text. Code page conversion will occur.
2. Info - The script will emit an Informational diagnostic message in the console log if a *non-printable* character is detected and the file is copied as text. Code page conversion will occur.
3. Warning - The script will emit a Warning diagnostic message in the console log if a *non-printable* character is detected and the file is copied as binary. No code page conversion will occur, and the offending characters will be treated like *non-roundtripable* characters.

It should be noted that for the “Do Not Check” and “Info” options, although code page conversion is taking place, this could cause an issue later. The file may not be easily maintained with a distributed editor, and you run the risk of having corrupted files.

Controlling what level of checking should be performed during the migration is done via the optional scan level `-np, --non-printable <level>` parameter switch passed to the migration script. If the scan level parameter is not specified, the *non-printable* characters are not checked (equates to the “Do Not Check” option).

5.3.4.1 Scan level set to info

```
$DBB_HOME/migration/bin/migrate.sh -r /u/user1/Migration -np info -m
MappingRule[hlq:MIGRATE.TECHDOC,extension:CPY,tolower:true,pdsEncoding:IBM-037]
"COPYBOOK(NPRINT)"

Non-printable scan level is info
Local GIT repository: /u/user1/Migration
Using mapping rule com.ibm.dbb.migration.MappingRule to migrate the data sets
Migrating data set COPYBOOK(NPRINT)
[INFO] Copying MIGRATE.TECHDOC.COPYBOOK(NPRINT) to /u/user1/Migration/copybook/nprint.cpy
using IBM-037
! Possible migration issue:
  Line 2 contains non-printable characters:
    Char 0x00 at column 19
  Line 3 contains non-printable characters:
    Char 0x06 at column 19
  Line 4 contains non-printable characters:
    Char 0x07 at column 19
  Line 5 contains non-printable characters:
    Char 0x1B at column 19

** Build finished
```

Note that the migration tool has detected numerous *non-printable* characters on various lines and has performed the copy as text and will be tagged on USS using the supplied encoding of IBM-037.

```
ls -alT /u/user1/Migration/copybook
total 64
          drwxr-xr-x  2 USER1  OMVS      8192 May  6 15:39 .
          drwxr-xr-x  4 USER1  OMVS      8192 May  6 14:55 ..
t IBM-037  T=on  -rw-r--r--  1 USER1  OMVS       114 May  6 15:39 nprint.cpy
```

5.3.4.2 Scan level set to warning

```
$DBB_HOME/migration/bin/migrate.sh -r /u/user1/Migration -np warning -m
MappingRule[h1q:MIGRATE.TECHDOC,extension:CPY,toLower:true,pdsEncoding:IBM-037]
"COPYBOOK(NPRINT)"

Non-printable scan level is warning
Local GIT repository: /u/user1/Migration
Using mapping rule com.ibm.dbb.migration.MappingRule to migrate the data sets
Migrating data set COPYBOOK(NPRINT)
[WARNING] Copying MIGRATE.TECHDOC.COPYBOOK(NPRINT) to /u/user1/Migration/copybook/nprint.cpy
! Possible migration issue:
  Line 2 contains non-printable characters:
    Char 0x06 at column 19
  Line 3 contains non-printable characters:
    Char 0x07 at column 19
  Line 4 contains non-printable characters:
    Char 0x1B at column 19

! Copying using BINARY mode
** Build finished
```

Note that the migration tool has detected numerous *non-printable* characters on various lines and has performed the copy as binary. The file remains untagged on USS (the manual tagging of the file is still required).

```
ls -alT /u/user1/Migration/copybook
total 48
          drwxr-xr-x  2 USER1  OMVS      8192 May  6 17:17 .
          drwxr-xr-x  4 USER1  OMVS      8192 May  6 17:17 ..
- untagged  T=off  -rw-r--r--  1 USER1  OMVS       320 May  6 17:17 nprint.cpy
```

5.4 Recommendations for using the Migration utility

A strategy must be decided on how to handle both *non-printable* and *non-roundtripable* characters found in source members that are to be migrated from z/OS PDSs to Git. The “HEXCoded” member, though not demonstrated in the above scenarios, is a common occurrence in many older legacy applications. Source code members such as this need to be identified and transformed to that shown in the “HEXVALUE” member if the goal is to manage the source seamlessly using the modernized tooling provided through Git. The section [“2.4 Managing non-roundtripable and non-printable characters”](#) provides more information.

The migration tool provides an option to perform a scan of the z/OS PDSs to assist in the analysis and correction/reconciliation of these situations prior to performing the copy to the local Git repository. Invoking this scan is done via the optional preview *-p*, *--preview* parameter switch and will bypass the copy. The default is “No Preview”.

```
$DBB_HOME/migration/bin/migrate.sh -r /u/user1/Migration -np warning -p -m
MappingRule[hql:MIGRATE.TECHDOC,extension:CPY,toLower:true,pdsEncoding:IBM-037] COPYBOOK

Non-printable scan level is warning
Preview flag is specified, no members will be copied to HFS
Local GIT repository: /u/user1/Migration
Using mapping rule com.ibm.dbb.migration.MappingRule to migrate the data sets
Migrating data set COPYBOOK
[WARNING] Previewing MIGRATE.TECHDOC.COPYBOOK(HEXCoded)
! Possible migration issue:
  Line 3 contains non-printable characters:
    Char 0x19 at column 37
  Line 4 contains non-roundtripable characters:
    Char 0x15 at column 37
  Line 5 contains non-printable characters:
    Char 0x0C at column 37
  Line 6 contains non-roundtripable characters:
    Char 0x0D at column 37

! Will copy using BINARY mode
Previewing MIGRATE.TECHDOC.COPYBOOK(HEXVALUE). Using IBM-037.
[WARNING] Previewing MIGRATE.TECHDOC.COPYBOOK(NPRINT)
! Possible migration issue:
  Line 2 contains non-printable characters:
    Char 0x06 at column 19
  Line 3 contains non-printable characters:
    Char 0x07 at column 19
  Line 4 contains non-printable characters:
    Char 0x1B at column 19

! Will copy using BINARY mode
[WARNING] Previewing MIGRATE.TECHDOC.COPYBOOK(NROUND)
! Possible migration issue:
  Line 2 contains non-roundtripable characters:
    Char 0x15 at column 27
  Line 3 contains non-roundtripable characters:
    Char 0x0D at column 27
  Line 4 contains non-roundtripable characters:
    Char 0x25 at column 27
  Line 5 contains non-printable characters:
    Char 0x0F at column 33
  Line 7 contains non-roundtripable characters:
    Empty Shift Out and Shift In at column 74

! Will copy using BINARY mode
```

With the `-l, --log` option, a log file can be created to contain all the messages about the migration process, including the `non-printable` and `non-roundtripable` characters encountered during the scan. This log file can be leveraged by the developers to perform the necessary changes in their original source code members prior to the real migration process.

Many other options of the Mapping Rule parameter can be leveraged to control the behavior of the Migration utility. These options are described on the IBM Documentation¹⁰ website.

¹⁰ <https://www.ibm.com/docs/en/adfz/dbb/1.1.0?topic=migrating-data-sets-git>

6 Conclusion

This document highlighted some pitfalls to avoid when migrating source code from z/OS PDSs to Git: correctly determining the original code page used when editing and reading source code members in z/OS is a key activity to ensure a smooth migration of these elements to Git. The second aspect is about managing the specific EBCDIC characters which are not easily converted to their UTF-8 counterparts. For these specific characters, known as *non-printable* and *non-roundtripable* characters, a decision must be taken to either refactor the source code to eliminate those characters, or transfer the files as binary. Both options have drawbacks that should be evaluated prior to the final migration to Git, as there is no easy way back.

The Migration utility shipped with IBM Dependency-Based Build (DBB) helps perform this migration activity, by automating the detection of *non-printable* and *non-roundtripable* characters, copying the files to USS, tagging them on USS and generating a `.gitattributes` file. Some scenarios are illustrated in this document, to facilitate the use of this utility.