Git 2.26.2 for z/OS Release Notes

This document provides tips for effectively using the Rocket Software z/OS port of Git 2.26.2. The reader should already have some familiarity with Git, z/OS, and z/OS Unix System Services (USS).

z/OS-specific changes since the previous (2.14.4) release

In addition to the platform-independent changes between Git release 2.14.5 and 2.26.2, the following changes apply to the z/OS port.

- Many more character encodings are now supported on z/OS
- The Git encoding (the character encoding used in the Git server repositories) for text will now always be UTF-8
- There are no longer 'default' encodings applied for files in the working tree. This means that, for any given file, if no treatment is specified in gitattributes file:
  - Git will not set a file tag during clone/checkout
  - Git will not allow a tagged file to be added to the index

Must-read: Migrating from Git for z/OS 2.3.5 or Git for z/OS 2.14.4

Setup

Prerequisite software

z/OS

z/OS 2.3 or higher is required for Git 2.26.2 for z/OS.

It is recommended that the fix for APAR OA52954 (ABEND0C4 RSN11 IN BPXPRECP WHEN COPYING EDSAMAGICPATH) be APPLYed.

z/OS Miniconda

Git 2.26.2 for z/OS (and all Rocket z/OS tool and language ports) are installed via conda and thus conda must first be installed on your system. In order to install conda, download and install z/OS Miniconda which contains all that is needed to install and run conda. z/OS Miniconda can be downloaded from the Rocket Customer Portal at http://my.rocketsoftware.com/.

Other Rocket open source products

Although Git 2.26.2 for z/OS does depend upon other open source languages (Perl) and tools (bash, curl, expat, libtag, openssl, zlib), these will be automatically installed when you install Git via conda. There is no need to manually download and install these dependencies.

Installation

Git 2.26.2 for z/OS is downloaded and installed via conda. Per the pre-requisites, you must already have conda installed in order to download and install z/OS Git.

Installation command:

```
conda install -c <channel> git
```

Using Git attributes

The previous versions of Git for z/OS introduced two concepts:

- `git-encoding`: The character encoding of files stored in the Git repository. These files are part of Git's internal database and are never directly edited by users.
- `zos-working-tree-encoding`: the character encoding of files stored in the working directory. These files are the reason Git exists. They are the user-defined content of the repository, and are what the user edits and compiles. This encoding is controlled via the Git attributes system.

> ⚠ **Note**: `git-encoding` attribute is no longer used in Git for z/OS 2.26.2
>
> Git 2.26.2 ignores `git-encoding` attribute and encodes the content from the specified encoding to UTF-8. To avoid misunderstanding it is recommended to remove all git-encoding attributes from .gitattributes.

If no attributes are set, Git for z/OS behaves essentially the same as it does on all other platforms: the encoding is never altered when files are copied between the working directory and the repository.

If no attributes are set, Git for z/OS will not set any tags on files in the working tree.

USS File tagging

z/OS Git relies upon the file tagging feature of USS. File tagging is used to identify the code set of text data within files.

These tags can be examined by using either the `ls` command with the `-T` option or the `chtag` command with the `-p` option. The following examples demonstrate the output of these commands for a directory in which file tags have been set:

```
bash-2.03$ ls -lT
total 1000
t ISO8859-1   T=on  -rw-r--r--   1 TSJLC    PDUSER      3240 Oct 20 14:22 Makefile
t ISO8859-1   T=on  -rw-r--r--   1 TSJLC    PDUSER       144 Oct 20 14:22 README.md
- untagged    T=off -rwxr-xr-x   1 TSJLC    PDUSER    372736 Oct 20 15:38 sysevent
t ISO8859-1   T=on  -rw-r--r--   1 TSJLC    PDUSER      1224 Oct 20 14:22 sysevent.cpp
m IBM-1047    T=off -rw-r--r--   1 TSJLC    PDUSER    126000 Oct 20 15:38 sysevent.o
bash-2.03$ chtag -p *
t ISO8859-1   T=on  Makefile
t ISO8859-1   T=on  README.md
- untagged    T=off sysevent
t ISO8859-1   T=on  sysevent.cpp
m IBM-1047    T=off sysevent.o
bash-2.03$
```

Most z/OS commands that manipulate text files (editors, such as `vi` and `emacs` 19.34, the XL C and C++ compilers, `make`, `grep`, etc.) will respect file tags, if present, and "do the right thing" with the text. If a file is not tagged, it is generally presumed to be encoded in IBM-1047 (EBCDIC).

The encoding tag on a file can be changed with using the chtag command, as follows:

```
chtag -t -c iso8859-1 <filename>   # to tag as ASCII
chtag -t -c ibm-1047 <filename>    # to tag as EBCDIC
chtag -t -c utf-8 <filename>       # to tag as UTF-8
```

Note: changing a file tag does not affect the binary content of a file.

Git and tags

When you add files to the index, Git verifies that the file tag corresponds to the `zos-working-tree-encoding` or `binary` attribute. If the file tag and attribute don't match, Git fails with one of the following errors:

```
fatal: can't add "<filename>": file is untagged, set correct file tag
fatal: can't add "<filename>": file tag (<file-tag-encoding>) does not match its attributes (<zos-working-tree-encoding>)
fatal: can't add "<filename>": file is tagged, set corresponding zos-working-tree-encoding attribute or reset file tag
```

When files in the working directory are updated by Git (via, for example, git checkout), they are converted and tagged according to `(zos-)working-tree-encoding` attribute using the following behavioral model.

Git attributes setting		Current file tag		Command behavior			
				git add		git checkout	
binary	**(zos-)working-tree-encoding**			**command status**	**conversion**	**command status**	**conversion**
specified*	regardless*	set	binary	ok	no conversion	ok	no conversion
			not binary	fails		ok, updates tag	
		not set		fails		ok, sets tag	
not specified	specified	set	match	ok	encoding ⇔ UTF-8	ok	UTF-8 ⇔ encoding
			doesn't match	fails		ok, updates tag	
		not set		fails		ok, sets tag	
not specified	not specified	set		fails		ok. resets tag	no conversion
		not set		ok	no conversion	ok, tag isn't set	

***Note**: if both working-tree-encoding and binary attributes are specified (and no zos-working-tree-encoding), the working-tree-encoding attribute will take precedence in Git for z/OS 2.26.2 and the file will be converted.

Supported encodings

A variety of encodings are now supported in Git for z/OS 2.26.2. If no attributes are set, the encoding is never altered when files are copied between the working directory and the repository. Otherwise, Git re-encodes the content from the encoding specified in zos-working-tree-encoding attribute to UTF-8 and back. The set of supported encodings is based on the z/OS iconv implementation. You can find a list of CCSIDs which are fully supported in `zos-working-tree-encoding` attribute in the Appendix A, "Supported character sets in Git for z/OS 2.26.2". Character encodings not listed in Appendix A may work for you (if they are fully supported by iconv) however they have not been tested and are thus not fully supported.

In heterogeneous deployments (e.g. both USS and Windows Git clients), the limitations of your zos-working-tree-encoding attribute choices must be respected across all Git clients. For example, given two Git client clones of a text file - one on USS using IBM-1047 encoding and one on Windows using UTF-8 encoding - you must not add an IBM-930 character to the same file on your Windows system.

.gitattributes examples

As noted, the Git attribute system is used to specify the encoding of files in both the Git repository and the working directory. Attributes are set by creating an attribute specification, which consists of a file-matching pattern (as specified when creating a .gitignore file) and one or more attributes. The attribute specification can reside in either the working directory (in a `.gitattributes` file) or in the Git repository (usually named `.git`), in the file `info/attributes`.

When stored in the working directory, the `.gitattributes` file can be committed to the repository like any other file, in which case the attribute specification will be automatically applied when the working directory is created via `git checkout`.

Some projects might not want to introduce a z/OS-specific file into the repository. In these cases, the attributes can be stored in the `.git/info/attributes` file.

Note: Attribute specifications, whether in `.gitattributes` or `.git/info/attributes`, **must** be encoded in ISO8859-1 or UTF-8.

A typical Git attributes file might look something like this:

```
# git's files (which MUST be ASCII or UTF-8)
.gitattributes   zos-working-tree-encoding=iso8859-1
.gitignore       zos-working-tree-encoding=iso8859-1

# The default for text files
*.cpp       zos-working-tree-encoding=ibm-1047
*.md                zos-working-tree-encoding=iso8859-1

# Binary files
*.jpg   binary
*.png   binary
*.gif   binary
*.zip   binary
```

Note: General usage specification of Git attributes files can be found here and specific documentation of binary option can be found here.

A Git attribute specification can be tested by using the `git check-attr` command:

```
bash-2.03$ git check-attr -a *
Makefile: zos-working-tree-encoding: iso8859-1
README.md: zos-working-tree-encoding: iso8859-1
sysevent.cpp: zos-working-tree-encoding: ibm-1047
*.png: binary: set
*.png: diff: unset
*.png: merge: unset
*.png: text: unset
bash-2.03$
```

Cloning an open source repository

The remote access protocols currently supported by Git for z/OS are `ssh` and `https`.

- To use `ssh`, you will need ssh access credentials on the remote server. In the case of github, accounts are available for free, and publicly-readable repositories do not require any further authorization to be cloneable.
- To use `https`, you generally do **not** need credential on the remote server. However, you **do** need credentials to push code to the remote server.

Here is an example of cloning a repository from github. If you have an account on github, this should work once you have installed Git for z/OS and configured the environment correctly.

```
bash-2.03$ git clone  -v git@github.com:zorts/hello_world.git
Cloning into 'hello_world'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 13 (delta 3), reused 13 (delta 3), pack-reused 0
Receiving objects: 100% (13/13), done.
Resolving deltas: 100% (3/3), done.
Checking connectivity... done.
bash-2.03$
```

This particular repository has a `.gitattributes` file in its top-level directory that will cause files to be checked out as EBCDIC.

Downloading a certificate bundle

The z/OS port of Git 2.26.2 supports the `https` Git remote access protocol in addition to the `ssh` protocol. To use this, you must set the environment variable `GIT_SSL_CAINFO` to point to a file containing the X.509 certificates of the public Certificate Authorities, in PEM format.

If you do not already have a suitable certificate file, you can download a current copy of the file from a trusted source and verify the signature of the file. A suggested source is the `curl` web site. If you have the Rocket ports of `curl` and `openssl` installed, you can use the following commands. These assume that:

- The path of the directory in which Git and the related tools were installed is in the environment variable RSUSR.
- You have write permission to that directory.
- You wish to store the certificate file in that directory.

```
# Make sure that there is an "etc" subdirectory in the Rocket ported tools directory
mkdir -p $RSUSR/etc
cd $RSUSR/etc

# Get the certificate file
curl -s -k https://curl.haxx.se/ca/cacert.pem -o cacert.pem

# Get the signature file and extract just the hash
curl -s -k https://curl.haxx.se/ca/cacert.pem.sha256 | awk ' {print $1}' > cacert.pem.sha256

# Generate the hash on the certificate file and compare it to the signature file.
# If the signature matches, there will be no output from diff.
openssl dgst -sha256 cacert.pem | awk ' {print $2}' | diff - cacert.pem.sha256
```

Once this has been done, you can set `GIT_SSL_CAINFO` to point to the file:

```
export GIT_SSL_CAINFO=$RSUSR/etc/cacert.pem
```

It is also possible for a Git user to disable the certificate checking by entering the following command. This is **not** recommended.

```
git config --global http.sslVerify false
```

Compatibility with previous releases

Migrating from working-tree-encoding to zos-working-tree-encoding

The initial release of Git for z/OS (version 2.3.5) used `working-tree-encoding` rather than `zos-working-tree-encoding` for controlling the encoding of files in the working tree. It is vital that such usage be migrated as soon as possible to the new attribute name.

Why is this necessary?

Release 2.18 of the platform-generic git code introduced a git attribute with the same name (`working-tree-encoding`) as the one used by the Rocket z/OS port of release 2.3.5. The meaning is essentially the same; however, this means that attempts to use a repository prepared on z/OS with other open source platforms (such as Windows, or the servers that typically run git servers such as GitHub or BitBucket) will now attempt to process this attribute, almost certainly with undesired results. It was considered unlikely that this attribute would ever appear in the platform-generic git; this turned out to be wrong.

On all platforms supported by Git 2.18+ `working-tree-encoding` forces conversion of the file between UTF-8 and the encoding specified in the attribute without respect to other attributes set for the file. With z/OS Git 2.26.2 where multiple uses of `working-tree-encoding` and `zos-working-tree-encoding` and `binary` are specified for the same file it is important to understand which take precedence:

- `binary` takes precedence over `zos-working-tree-encoding`
- `zos-working-tree-encoding` takes precedence over `working-tree-encoding`
- `working-tree-encoding` takes precedence over `binary`

Because this is a circular order of precedence, it is vital you not use all three for the same file or unpredictable results will ensue.

What action is required?

Git repositories on z/OS that use the `working-tree-encoding` attribute should be altered immediately to use the new `zos-working-tree-encoding` attribute.

The `working-tree-encoding` attribute will continue to be honored on z/OS; however, as time goes on and git on other platforms is updated to 2.18 and beyond, the likelihood of problems when using GitHub, BitBucket, and Windows git-based tools will increase. When processed by git 2.26.2, the `zos-working-tree-encoding` attribute takes priority over `working-tree-encoding`.

References

- Background discussion on why working-tree-encoding was added to the platform-generic code
- The first report of the conflict involving the z/OS attribute

Migrating from Git for z/OS 2.3.5 or Git for z/OS 2.14.4

If you have been working with previous releases Git for z/OS, you must push recent changes to the remote repository with the old version of Git and clone the repository with Git for z/OS 2.26.2.

> ⚠ **Note: Once you change a repository with Git for z/OS 2.26.2, this repository won't be compatible with previous versions of Git for z/OS. It is vital that entire team will move to the new version of Git.**

When you clone the repository with Git for z/OS 2.26.2, you need to run `git status` command. You might see that Git marks some files with the attribute as `modified`, even if you didn't modify them.

Usually it happens if you work in the repository, which had files encoded in ISO8859-1 in the index and the files have bytes which are converted to bytes greater than x7F in ISO88859-1. This is an expected behavior. Git for z/OS 2.26.2 does real-time conversion to UTF-8 to check if there is a difference between a file in the repository index and working directory. The files used to be encoded in ISO8859-1 in the repository index. If the ISO8859-1 representation doesn't match that in UTF-8, the file will be marked as modified. We recommend to update the files in the index (commands: `git add <filename>; git commit`).

Also it can happen if the file in the index has a character which doesn't have a corresponding character in the encoding defined in (`zos-`)`working-tree-encoding` attribute. Note that iconv on z/OS replaces such characters with a substitution character without any warning or error message. You need to exclude such characters from the file or change (`zos-`)`working-tree-encoding` attribute.

Restrictions

- Some files used by Git **must** be encoded as ISO8859-1 or UTF-8. These include:
  - Git attribute files, whether in `.git/info/attributes` or `.gitattributes`
  - `.gitignore` files
- The only remote protocols supported are `ssh` and `https`.
- Only client mode is supported; in other words, Git for z/OS can clone from, and push to, remote repositories via `ssh or https`, but cannot be the target of clone and push from other clients.
- In heterogeneous deployments (e.g. both USS and Windows Git clients), the limitations of your zos-working-tree-encoding attribute choices must be respected across all Git clients. For example, given two Git client clones of a text file - one on USS using IBM-1047 encoding and one on Windows using UTF-8 encoding - you must not add an IBM-930 character to the same file on your Windows system.
- The Git interface to Subversion (git-svn) is not supported.

Known issues

- Very large repositories (especially repositories with a lot of history) may cause clone and checkout operations to fail with this symptom:

  ```
  fatal: Out of memory? mmap failed: EDC5124I Too many open files. (errno2=0x07360344)
  ```

  The solution is to restrict Git to using less memory, by setting these configuration variables:

  ```
  pack.packSizeLimit 20m
  core.packedGitWindowSize 16m
  core.packedGitLimit 32m
  pack.windowMemory 32m
  pack.thread 1
  pack.deltaCacheSize 1m
  ```
- Git for z/OS relies on the z/OS file tagging facility. For this reason, Git repositories on z/OS **must** reside in file systems that support that facility. Notably, a file system mount via NFS from a non-z/OS system cannot be used.

Appendix A - Supported character sets in Git for z/OS 2.26.2

#	Character set	#	Character set	#	Character set
1.	IBM-037	23.	IBM-935	45.	IBM-1156
2.	IBM-273	24.	IBM-939	46	IBM-1157
3.	IBM-274	25.	IBM-1025	47.	IBM-1158
4.	IBM-275	26.	IBM-1027	48.	IBM-1165
5.	IBM-277	27.	IBM-1047	49.	IBM-1364
6.	IBM-278	28.	IBM-1112	50.	IBM-1390
7.	IBM-280	29.	IBM-1122	51.	IBM-1399
8.	IBM-282	30.	IBM-1123	52.	IBM-4971

9.	IBM-284	31.	IBM-1124	53.	IBM-5123
10.	IBM-285	32.	IBM-1140	54.	IBM-8482
11.	IBM-297	33.	IBM-1141	55.	IBM12712
12.	IBM-424	34.	IBM-1142	56.	BIG5
13.	IBM-425	35.	IBM-1143	57.	ISO8859-1
14.	IBM-500	36.	IBM-1144	58.	ISO8859-2
15.	IBM-870	37.	IBM-1145	59.	ISO8859-5
16.	IBM-871	38.	IBM-1146	60.	ISO8859-7
17.	IBM-875	39.	IBM-1147	61.	ISO8859-8
18.	IBM-901	40.	IBM-1148	62.	ISO8859-9
19.	IBM-921	41.	IBM-1149	63.	TIS-620
20.	IBM-923	42.	IBM-1153	64.	UTF-8
21.	IBM-924	43.	IBM-1154		
22.	IBM-933	44.	IBM-1155		