Bash Script – General Orientation
Updated October 2018
Modified for Russell OHM from original documentation by Adriane Hanson (Head of Digital Stewardship, UGA Libraries).

General overview:

The script runs system commands, command line tools, and other software programs via the command line to automate the process of preparing a batch of Archival Information Packages (AIPs).

The script is compatible with Mac or Linux.

Most actions are taken on folders/files within the current directory that the script is running on. Filepaths that point to documents/scripts which are external to the source directory (e.g. stylesheets, prepare_bag script) can be updated by changing the filepaths for the global variables at the beginning of the script.

Scripts we are using: aip-creation.sh

- for more detailed information about what the script does, review the document "aip-creation-script-overview" in the documentation folder on GitHub

<u>Common components of the scripts:</u>

For loops
The commands inside the loop are done to each of the folders or files selected by that loop. Usually this is the aip directory, although it can be a specific document type such as MediaInfo XML.

Loops may contain commands for running a tool (e.g. bagit.py) and/or bash commands (e.g. mkdir). One loop may contain multiple commands which are run in sequence on the selected items. Loops may also contain if-tests to only act on a portion of the selected items. General structure:

```
for d in x; do for each x (aka file/directory/etc) and is given the variable name d, do the following:
    command first this command
    command then this command
    etc.
done ends the loop
```

If/Then/Else
Tests if a condition is true before applying a command. Can test for multiple conditions at once which can be combined using OR (when only one must be true) or AND (when all must be true). "If" is always used with "then" (what to do if the test is true) and can optionally include "else" (what to do if the test is not true). General structure:

```
if [ test ] test for a condition, e.g. *_master.xml for if the file name ends with _master.xml
    then if the test is true...
        command … this is what the script should do
    else  if the test is not true… (optional - without else, it will do nothing if the test is false)
        command...this is what the script should do
fi  end of the if test ("if" spelled backwards)
```

Variables:
Variables refer to some value by a name while running commands, such as the aip directory or the output from a tool. The script defines variables in two ways:

- In for-loops, variables are set in the first line. The syntax is **for d in x; do** where d is the variable name and x is the variable value. Example: **for d in *; do** sets the variable d to mean each item in the current folder.

- For master.xml and bag validations, a variable is set to mean the output from the validation tools. The syntax is **valid=$((tool command) 2>&1)** where valid is the variable name and everything inside $() is the variable value. The variable's value is a combination of the tool's error output and the message it would normally display in the terminal after running.

We use the naming convention of the letter "d" for a variable that is for directories and the letter "i" for files (items). Variables are referred to in the commands as `"$d"` or `"$i".` The $ indicates it is a variable and the quotes make sure any spaces in the folder or file name will be treated properly in the terminal.

Arguments:
Arguments refer to information put into the terminal after the script name when the script is run. These are used to supply outside information into the script. Arguments are referred to in commands based on their order after the script name in the terminal: the first is `"$1"` and the second is `"$2".` In our scripts, `"$1"` is the source directory, used to navigate the file directory.

Standard bash commands

- **#** indicates that row is a comment. Used to explain what each step does. Can also be used to temporarily deactivate a command instead of deleting it when testing the script
- **cd path/to/directory** change directories; becomes the directory that relative filepaths in the script start from, although the current directory displayed in the terminal won't change.
- **cp what/to/copy where/to/copy/it/to** Copies items from one place to another.
- **echo "text"** displays the text in the terminal. Used to display the script's status and error messages.
- **mkdir directory-name** makes a folder inside the current directory with that name
- **mkdir "$d"/directory-name** makes a folder inside each "$d", with "$d" being defined by the for loop
- **mv what/to/move where/it/goes** move something to a new location; also used to rename files/folders
- **rm path/to/file** deletes the file
- **rmdir path/to/folder** deletes the folder - must be empty (otherwise use: **rm -f path/to-folder**)

Checking for existence of a file/directory before doing the command:

Note: according to our naming conventions, folders containing digital preservation files should have names beginning with the letter combination 'rbrl'

Tests if aip-directory whose name begins with 'rbrl' exists before running a command a command on the directory. Testing for existence reduces the amount of errors that the command line will throw when it cannot find the file to execute a command on. Additionally, because many commands depend on the output of previous commands as their input, if an error occurred earlier in the script, then it will recursively cause errors in the rest of the script that was dependent on an earlier piece of information.

```
for d in rbrl*; do
    if [ -d "$d" ]  if a directory beginning with 'rbrl' exists in the source directory
        then
            command   does this only when the folder actually exists
    fi
done
```

Naming:

When an output file, such as the master.xml file, is made as part of the script, the value of the variable from the for loop can be used as part of the filename with the syntax ${d} or ${i}. This is most commonly done to include the aip-id as part of the filename. Just a portion of the variable value can be used and new things can also be added. Whatever is in the curly braces { } is kept in the renaming. If a percent sign (%) is included in the curly braces, whatever comes after the % is deleted when renaming.

Command	**Value of variable**	**Resulting filename**
${d}_mediainfo-cleanup.xml	aip-id	aip-id_mediainfo-cleanup.xml
${i%_mediainfo-cleanup.xml}_master.xml	aip-id_mediainfo-cleanup.xml	aip-id_master.xml

Renaming:

The mv command is also used for renaming by "moving" a file to the same location but with a different name. This can be done with find/replace syntax to change or remove a portion of the existing name. Additional characters can also be added to the existing name.

Value of variable	**Command**	**Resulting filename**
aip-id	mv "$d" "${d}_bag" *(adds _bag to the end of the variable name)*	aip-id_bag
aip-id_bag	mv “$d” “${d%_bag} *(removes _bag from the end of the variable name)*	aip-id
N/A – not a file, but rather renames subdirectories	mv "$d/data" "$d/objects" *(renames directory from data to objects)*	n/a