This is a reference for the article authoring process, which is comprised of writing content and submitting it to version control.

It will not fully explain everything, and is subject to change over time.

The version control we (and a majority of the world) use is based on the command line tool 'Git', however we recommend using the GitHub Desktop client. Consult the *Authoring Environment Setup Guide* for instructions on installing the desktop client and connecting it to the correct Bitbucket repository.

If you've ever made a Publish in Settings Editor, you've followed the version control process; sans editorial review.

#### Analogous Tasks:

Select Repository ---> Select location

Write or edit article ---> Create new or edited settings

Commit the change ---> Save

Create a branch ---> Name the Changeset

Commit message ---> Changeset details

Merge branch ---> Publish into the real world where end-users can see

Merge Conflict ---> Changeset Error

#### Absent from Settings Editor:

Pull Request ---> Peer/SME Review prior to publishing

# The Author Workflow

## Process in brief

1. Access the Bitbucket Online Repository.
2. Create a "branch" for your article(s)
3. Copy (or clone) the repository to your local machine - This is only done once.
4. Add a file to your local copy of the repository.
5. "Commit" (save) the changes. This saves to your local copy of the repo only.
6. "Push" (send) your changes to your articles branch on Bitbucket Online.
7. Open a "pull request" (propose changes to the Master branch).
8. The approver will "Merge" your branch to the Master branch.
9. "Pull" (download) the updated version of Master to your local machine (so your local copy of Master has the changes you just made to the online copy).

## Organize/Prepare

* Make sure you have access to the repository and have setup your authoring environment as per the authoring environment setup guide.
* Save your pictures in 'images' and content in 'articles':
* /articles
  + SyncFailures.md
* /images
  + SyncFailureMessage.png

## Branch off the Master Repository

* Create a branch for your article, such as "brink4.1eWhatsNew-MN".
* Name it something rational. Initials at the end will help as we collaborate.
* "printerInstall" and "printerInstall" could be two different authors covering two different models.
* Like publishing Changesets and for the same reasons, "small, often, related" is advised.

## Write

* Using GitHub Flavored Markdown (GFM) syntax.
* Refer to the [GFM Cheat Sheet](https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf) as necessary.
* Using your choice of text-editor.
* See editor Recommendations.
* Save the text files with a '.md' or '.markdown' extension.

## Update your local copy of the branch

* "Fetch" or "Pull" in any changes the online copy of the branch you are working on may have.
* Stage the files you are going to commit to your local repo.

## Commit your changes

* Commit the changes or new files to your local repo.
* The Commit Message is like 'Details' when publishing a Changeset.

## Update the remote copy of the branch

* Select the correct branch
* Push changes to the branch

## Send your changes to the Master repository

* Create a Pull Request in Bitbucket.

## Merge Article Branch Into Master

* Those with a 'Publisher' role who are responsible for reviewing content will approve and merge it into the Master branch. This will not be every author.

## Update your local copy of the Master branch

* Start/End the cycle.
* You can't pull or refresh too much. You won't lose your work, and any changes won't overwrite yours simply through a pull request.

## That's it?

Technical details of the rest of the publishing pipeline are abstracted for the benefit of contributors, but all are welcome to learn. Contact Training & Documentation for more information.

## Best Practices

* Double check the branch you are committing changes to.
* Prevent merge conflicts by pulling/fetching the latest status of a branch before committing or pushing.
* Never commit binary files (such as an .exe) or output generated from the source files.
* Make each commit a logical unit.
* Share your changes with colleagues after completing a logical unit of work. (this also means it will get built into the KB when a build runs.)
* Use descriptive commit messages.
* Avoid indiscriminate commits.
* Pull/Fetch frequently.
* Create a Pull Request only once you believe your work is ready to be incorporated into the projects main repository.