Understanding Git and GitHub Workflow

Terminology

- Fork – a clone of a repository hosted on a Git server. We use the GitHub server. You only need to fork a repo once in your GitHub account.
- Origin – an alias for your fork on GitHub
- Upstream – an alias you create for the source of your fork (the original Microsoft repository)
- Clone – a copy of a repository on your local machine. This should be a copy of your Fork. You only need to clone your fork once per PC you use it on.
- Branch (Working Branch) – a logical workspace for changing content within your local clone
- Working directory – a physical workspace on disk containing your content files and folders
- Pull – the operation to update your local repository with latest version from a remote repository (fetch & merge). In our case, the remote repository will always be the upstream repository.
- Push – the operation to write the changes you made back into a remote repository. In our case, the remote repository will always be the origin repository (your fork).
- Fetch – gets the latest version of the files and changes that you do not have locally
- Merge – merges the current changes into your local repository
- Index – Git metadata used to track files and the git objects that represent the changes. The Add command adds files to the index so that changes can be tracked.
- Object store – Git metadata containing the four git objects (blob, tree, commit, and tag)

GitHub is not Git

GitHub is just a server for hosting repositories. Anyone could set up a git server. Setting up a git server is covered in Chapter 4 of the *Pro Git* book. There are other hosted git services available on the internet (BitBucket, Codeplex, etc.).

GitHub hierarchy

- Organization/Account (examples: Azure, Microsoft, MicrosoftDocs, PowerShell)
    - Repository (example: azure-docs-pr, SystemCenterDocs-pr,)
        - Branch (example: master)
    - Fork – a clone of a repository in your GitHub account
        - Branch (example: master)
        - Branch (example: July2016Freshness)

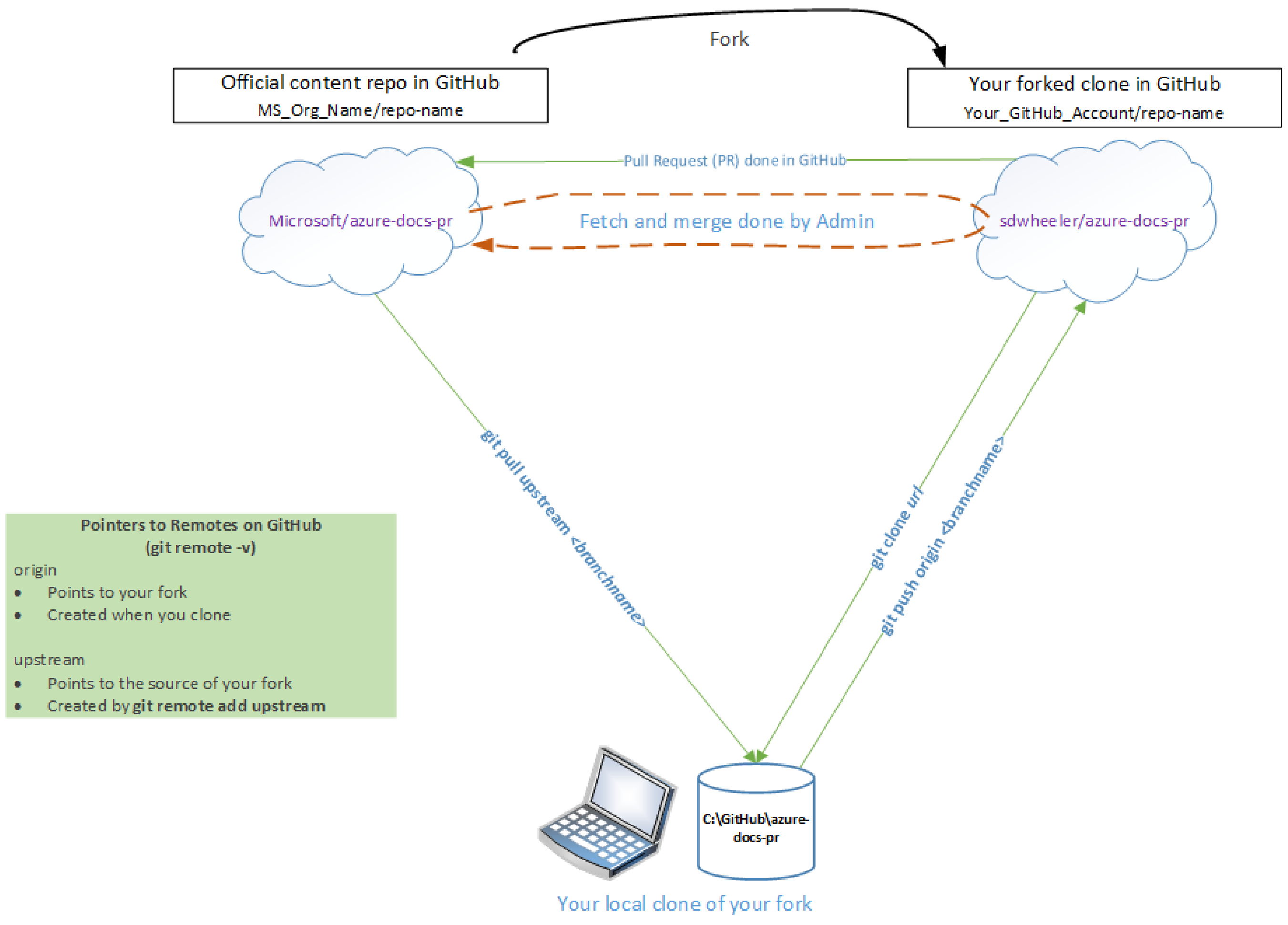
Forks and why you need one

- A Fork is a clone of a repository hosted on GitHub in your personal account.
    - Your fork is also, yet another backup of the main repository. This is a key feature of a distributed version control system.
    - If your local disk crashed causing you to lose your local repo, you can always clone your fork to another computer and work from there.
- You do not have rights to write (push) to the official repository. You must send a Pull Request. Then the admins of the official repository will fetch the branch from your fork and merge it into the master branch of the official repository. This protects the official repository as the source of truth for all content.
- You are not running a git service. GitHub cannot pull from the clone on your local machine. You must push your changed into your remote fork on GitHub.

Branches and why you need them

- Git stores data as a collection of snapshots that contain the changes you made. A Branch is a named label for that snapshot collection.
    - When you commit your changes, Git stores a commit object that contains a pointer to the snapshot of the staged content, the author, and the description of the commit.
    - Creating a new branch gives you a new working context within Git to make your changes without affecting the master branch.
    - Later, your working branch can be merged back into master, deleted, or kept indefinitely as a separate release path.
- A branch is **NOT** a folder on your local file system.
    - When you check out a branch, Git changes the files in the file system to match the versions in that branch's snapshot.
    - Git allows you to switch branches, safely, without losing any of the work you had done.
    - If you switch branches, the current state of the branch is stashed in the Git object store and the files on disk are changed to match the state of the new branch you switched to. As a result, if you check out different branches, you can literally watch the file system change as Git changes it to match the state of the branch.
- What is a **tracking branch**?
    - The `master` branch is created as a tracking branch for `origin/master` when you clone a repo.
    - You can create a tracking branch using the following command:
      `git checkout -b <branch> -t <remotename/branch>`
    - When you have a tracking branch set up, `git pull` will look up what server and branch your current branch is tracking, `fetch` from that server and then try to `merge` that into your local branch.

Understanding GitHub Workflow


Fork
Official content repo in GitHub
MS_Org_Name/repo-name
Your forked clone in GitHub
Your_GitHub_Account/repo-name
Pull Request (PR) done in GitHub
Microsoft/azure-docs-pr
Fetch and merge done by Admin
sdwheeler/azure-docs-pr
git pull upstream <branchname>
git clone url
git push origin <branchname>
C:\GitHub\azure-docs-pr
Your local clone of your fork

Pointers to Remotes on GitHub
(git remote -v)

origin
- Points to your fork
- Created when you clone

upstream
- Points to the source of your fork
- Created by **git remote add upstream**

Git Object Model

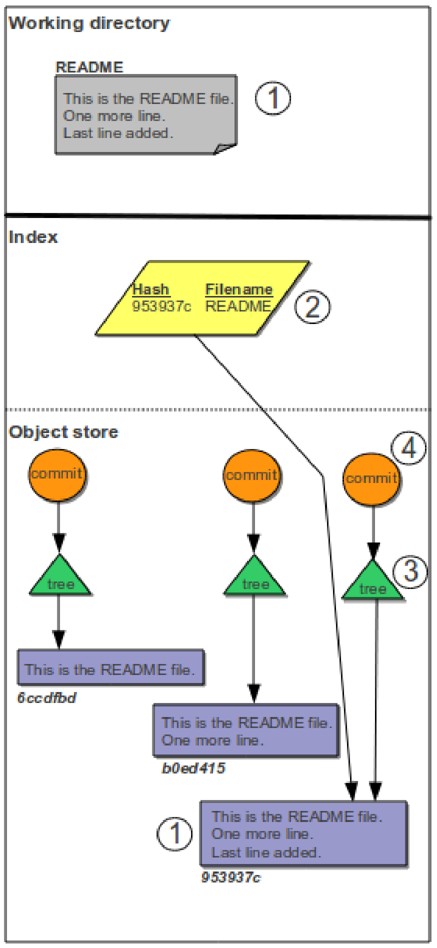
A git repository is defined by the data stored in the hidden .git folder on the local file system in the root folder of the repository. Git tracks the state of the repository in a database called 'index' and collection of files and folders known as the git object store.

Git Object Types

Blob object	The git "blob" type is just a bunch of bytes that could be anything, like a text file, source code, or a picture, etc.
Tree object	A git tree is like a filesystem **directory**. A git tree can point to, or include: 1. Git "blob" objects (similar to a filesystem directory includes filesystem files). 2. Other git trees (similar to a filesystem directory can have subdirectories).
Commit object	A git commit object includes: • Information about who committed the change/check-in/commit. For example, it stores the name and email address. • A pointer to the git tree object that represents the git repository when the commit was done • The **parent** commit to this commit (so we can easily find out the situation at the previous commit).
Tag object	A git tag object points to any git commit object. A git tag can be used to refer to a specific tree, rather than having to remember or use the hash of the tree.

Git objects in action

The following picture labeled “Diagram 9” is a view of the file system and the git index and object store. This example shows the state of the repository after several changes and three commits. Notice that the working directory contains only one file while the object store contains three blobs representing the contents of each version of README that was committed.



Session

Diagram 9

```
$ echo This is the README file. > README
$ git add README
$ git commit -m “Initial commit.”
$ echo One more line. >> README
$ git commit
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#     modified:   README
#
no changes added to commit (use "git add" and/or "git commit -a")
$ git add .
$ git commit -m “Added a second line.”
$ echo Last line added. >> README
$ git commit -a -m”Final commit.”
```

Result

This time, we skipped the “**git add**” command and went right to “**git commit -a ...**”.

The “**-a**” option causes “**git commit**” to compare every filename listed in the index with the same filename in the working directory. If the file in the working directory has been changed, git will:

* Copy the changed file from the working directory into a blob in the object store.
* In the index, change that file's hash to the hash of the newly create blob.

1 The updated contents of the **README** file is copied to the git object store and saved as a “blob”. The blob's hash is **953937c**.

2 Updated the index: Filename **README** now has a new hash in the git object store: **953937c**.

3 Created a new tree in the object store. The tree will contain a reference to objects in the index: The object store hash of the updated **README** file (**953937c**) and the filename (**README**).

4 Created a new git commit object. The commit object will reference the tree created in the previous step.

Setting up your working environment

Follow the instructions for setting up the tools as described in the Azure Contributor Guide Tools and Setup document for the following tasks:

- Creating a GitHub account and setting up your profile
- Creating LiveFyre account
- Configuring permissions in GitHub
- Setting up two-factor authentication

The document also includes instructions for setting up the Git client and a markdown editor. Those instructions are accurate and valid but I recommend the following changes:

- Install the GitHub Desktop client for Windows
- Install Visual Studio Code as your markdown editor

Install the Git for Windows and Posh-Git

Follow the instructions to install Git for Windows as I have outlined in my blog at:
https://seanonit.wordpress.com/2016/12/05/using-git-from-powershell

These instructions enable you to use Git from PowerShell. I also include instructions to setup a Git-enabled command prompt and to configure Git settings. Following these instructions will install the Windows Credential Manager for Git. Using the Windows Credential Manager means that you don't have to provide your Git username and token in the upstream URL.

Install Visual Studio Code as your markdown editor

Visual Studio Code is a lightweight but powerful source code editor which runs on your desktop and is available for Windows, OS X and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages (C++, C#, Python, PHP) and runtimes.

VS Code ships monthly releases and supports auto-update when a new release is available. If you're prompted by VS Code, accept the newest update and it will be installed (you won't need to do anything else to get the latest bits).

The benefits of using VS Code are the availability of extensions powerful extensions and the wide support of a growing community of users. Being a Microsoft open source project means that we have unique access to the project owners.

Installation

1. Download the Visual Studio Code installer for Windows.
2. Once it is downloaded, run the installer (VSCodeSetup-stable.exe). This will only take a minute.
3. By default, VS Code is installed under C:\Program Files (x86)\Microsoft VS Code for a 64-bit machine.

VS Code Extensions

I recommend installing the following extensions for the best user experience when using VS Code. VS Code has an internal command interface that is used to install extensions. To install an extension, launch VS Code Quick Open (Ctrl+P), enter the install command, and press enter. You need to restart VS Code for the new extensions to be loaded. However, to save time, you can install all of these extensions then restart VS Code only once after all extensions have been installed.

Mardown-oriented Extensions
Extension: markdownlint **Install command:** ext install vscode-markdownlint **Description:** markdownlint includes a library of more than 40 rules to encourage standards and consistency for Markdown files. This helps you avoid rendering problems in staging.
Extension: Markdown Shortcuts **Install command:** ext install markdown-shortcuts **Description:** Allows you to use shortcuts to edit Markdown (.md, .markdown) files. Add hotkeys for bold, italics, code blocks, bullets, numbered lists, and easy hyperlink creation.
Extension: Code Spellchecker **Install command:** ext install code-spell-checker **Description:** Load up a file and get highlights and hovers for spelling and grammar issues. Checking will occur as you type. The extension will offer spelling and grammar suggestions when you hover over the problem text.
Extension: Reflow paragraph **Install command:** ext install reflow-paragraph **Description:** Format the current paragraph to have lines no longer than your preferred line length, using alt+q (may be overriden in user-specific keyboard-bindings.) This extension defaults to reflowing lines to be no more than 80 characters long. The preferred line length may be overriden using the config value of reflow.preferredLineLength. By default, preserves indent for paragraph, when reflowing. This behavior may be switched off, by setting the configuration option reflow.preserveIndent to false.
Extension: Acrolinx for APEX **Install command:** See https://review.docs.microsoft.com/en-us/help/contribute/contribute-acrolinx-vscode?branch=master **Description:** Acrolinx is software that provides content authors with automated feedback on grammar, spelling, punctuation, writing style, terminology, and voice. Acrolinx is available both upstream and locally - upstream, users get automatic results from the Acrolinx integration for GitHub, which writes Acrolinx results to each pull request. The tool is seamlessly integrated into the pull request workflow. Locally, the Acrolinx extension for Visual Studio Code is now available so you can obtain the Acrolinx feedback before you push content to the upstream repository.
Extension: Gauntlet Authoring Services and VS Code Extension **Install command:** See https://review.docs.microsoft.com/en-us/help/contribute/contribute-vscode-extension?&branch=master **Description:** The Gauntlet VS Code extension for OPS authoring provides Markdown authoring assistance to writers working in OPS and publishing to docs.microsoft.com. It includes several functions, including applying templates to new Markdown files, applying common formatting to strings, and inserting links, images, tokens, snippets, tables, and lists, as well as previewing content using your site's CSS.

Extension: Replace Smart Characters **Install command:** ext install DrMattSm.replace-smart-characters **Description:** This extension replaces those pesky "smart" characters from Word (and also some fancy HTML characters) with their more common and friendly counterparts.

Language-oriented Extensions
Extension: C# for Visual Studio Code **Install command:** ext install csharp **Description:** The C# extension for Visual Studio Code provides the following features inside VS Code: • Lightweight development tools for .NET Core. • Great C# editing support, including Syntax Highlighting, IntelliSense, Go to Definition, Find All References, etc. • Debugging support for .NET Core (CoreCLR). NOTE: Mono and Desktop CLR debugging is not supported. • Support for project.json and csproj projects on Windows, macOS and Linux.
Extension: JS-CSS-HTML Formatter **Install command:** ext install vscode-JS-CSS-HTML-formatter **Description:** This extension wraps js-beautify to format your JS, CSS, HTML, JSON file.
Extension: PowerShell Language Support for Visual Studio Code **Install command:** ext install PowerShell **Description:** This extension provides rich PowerShell language support for Visual Studio Code. Now you can write and debug PowerShell scripts using the excellent IDE-like interface that Visual Studio Code provides.
Extension: XML Formatter **Install command:** ext install vs-code-xml-format **Description:** A simple wrapper around https://github.com/FabianLauer/tsxml/ for formatting XML in VS Code. Currently, only complete documents can be formatted. Formatting selections is planned.

Git Workflow Tasks

This sections describe several common tasks you will perform to accomplish work.

One-time setup for contributing to a new repository

Commands and actions	What happens and why
1. Fork the repository in GitHub Log into GitHub and navigate to the private repository. Go to the top-right of the page and click the Fork button. If prompted, select your account as the location where the fork should be created.	This creates a copy of the repository within your Git Hub account. Generally speaking, technical writers and program managers need to fork the private repo (azure-docs-pr or azure-docs-powershell). Community contributors need to fork the public repo.

2. Clone your fork to your local machine
 	
Open your Git Shell and run the following commands:
`cd C:\github`
`git clone https://github.com/<your GitHub user name>/azure-docs-pr.git`
	This copies you fork of the official repository to your local machine. A files are downloaded and the master branch is checked out automatically. Also the 'origin' alias is created automatically to refer to your remote fork on GitHub.
	
	In this example, your git repositories are contained in C:\github on the local disk.

3. Create the upstream reference to the official source repository.
 	
Run the following command from your Git Shell:
`cd C:\github\azure-docs-pr`
`git remote add upstream https://github.com/Azure/azure-docs-pr.git`
	This creates the 'upstream' alias for the remote private repository on GitHub. There is nothing special about the name 'upstream'. This is just a common practice. All Git documentation will use this name to refer to the repository that is the source of your fork.

Notes
These tasks only need to be done once for a given repository. Once you have forked the repository you can clone it to as many machines as you want. The fork is a cloud-based backup of your work. If your local hard drive crashed, you could clone your fork to a new machine. You will only have lost any changes that were not pushed into your fork.

Normal editing workflow

Commands and actions	What happens and why

1. Create a new working branch
 	
`cd C:\github\azure-docs-pr`
`git pull upstream master:newbranch`
	This will pull the latest contents from the upstream remote and create a new branch named 'newbranch'.
	
	You can skip this step if you are returning to continue work on the same branch.

2. Check out the working branch `git checkout newbranch`	This tells git to switch to the working branch context. The command prompt in the Git Shell should show this branch name. Git also updates the files on disk to match the state of this branch.
3. Make additions and changes to your content.	This is done using your content editing and creation tools like VS Code or Atom.

4. Add your changes to Git's tracking database.
 	
`git status`
`git add --all`
	Git keeps an index of all of the files that are being tracked. When you add or change files in the repository you need to update the Git index. The `status` command will show you which files are being tracked and which are not. The `add` command adds files to the index. If a file is not being tracked, it cannot be committed to the repository.

5. Commit your changes. `git commit -m "description of the changes"`	This checks-in the changes to your local git repository.
6. Pull the upstream master into your working branch again. `git pull upstream master`	While you were working, the upstream repository could have changed. Other contributors could have checked-in updates that you do not have synced to your local repository. The `pull` command ensures that your branch contains the latest version of the content. You may now have conflicts that need to be resolved. If so, fix the conflicts and commit the changes again.
7. Push your changes to your fork. `git push origin newbranch`	Now your fork is in sync with your local repository. You are ready to send a pull request to have your changes merged into the official repository.
8. Submit a pull request. Log into GitHub and navigate to your fork. You should see that new commits have been added. There will be a button to create a pull request. Click that button, review your changes, and submit.	Unless you are an Admin for the repository you do not have write permissions. So you cannot push changes into the official repository. You must create a Pull Request (PR). An Admin for the repository will review your request. If there are no validation errors or other problems, the Admin will pull the changes from your fork and merge them into the master branch of the official repository.
Notes	

Working in release branch

Notes	
Working in a release branch is the same as your daily work flow. The only change is that you want to create a tracking branch. When you have a tracking branch set up, git pull will look up what server and branch your current branch is tracking, fetch from that server and then try to merge that into your local branch.	
Commands and actions	**What happens and why**
1. Fetch the latest list of branches `git fetch upstream`	This fetches a list of the branches from the upstream repo. Before running this command, your local repo does not know about the release branch
2. Create a new tracking branch `git checkout -b <branch> -t <upstream/branch>`	This command creates a new branch, checks it out, and links it for tracking to the remote branch in the upstream repo.
3. Pull the upstream branch into local branch. `git pull upstream`	This copies the latest content in release branch down into your local branch.
4. Make additions and changes to your content.	This is done using your content editing and creation tools like VS Code or Atom.

5. Add and commit your changes.
 	
`git add --all`
`git commit -m "description of the changes"`	

6. Push your changes up into your fork. `git push origin branch`	Now your fork is in sync with your local repository. You are ready to send a pull request to have your changes merged into the official repository.
7. Submit a pull request. Log into GitHub and navigate to your fork. Be sure to select the release branch in the GitHub UI. There will be a button to create a pull request. Click that button, review your changes, and submit.	Your PR will be processed and merged into the release branch of the official repo. When the release goes live, the PR admins will merge the release branch into the master branch.

Throw away an uncommitted branch and start over

Commands and actions	What happens and why
1. git reset --hard	This resets all files that have changed since the last commit. This is a way to undo your changes and get back to a known state.
Notes	

Keeping your repos in sync

Commands and actions	**What happens and why**

1. Pull the upstream master into your working branch again.
 	
Open Git Shell
`cd C:\github`
`git checkout master`
`cd C:\github\azure-docs-pr`
`git pull upstream master`
	The `checkout` command ensures that you are in the master branch of your local repository. The `pull` command copies the current version of the master branch from the upstream remote into the currently selected branch (master).

2. Push the local master branch into your fork. `git push origin master`	The `push` command uploads the current state of your local repository into your fork on GitHub.
Notes	
While this is not required, it is recommended as a best practice to keep your local repository and your remote fork in sync with the official source repository. This is a good practice to do if you have been away from working in a repository for any extended period of time.	

Deleting a branch

Commands and actions	**What happens and why**
1. Delete the local branch. `git branch -d branchName`	This prevents it from being accidentally pushed later. If the branch has unmerged changes git will warn you and will not delete the branch.

2. Delete the remote tracking branch.
 	
`git show-branch -r`
`git branch -dr upstream\branchName`
	Depending on how you check out a branch there may be a remote tracking branch. This happens automatically for 'master' when you clone. The `show-branch` command shows you all of the remote tracking branches.

3. Delete branch from your fork. `git push origin --delete branchName`	This updates your fork by telling it to delete the branch from the repository in GitHub.
Notes	
Branches should be deleted after they are merged into the official repository. This prevents the visual clutter of a long list of branches in your repository. These branches also get propagated to all forks of the repository.	

Restore a file from a previous commit

Notes	
For this scenario, you need to recover an older version of a file that was committed. For example, I did a bulk change on 300+ articles. One of the articles I changed overwrote the changes that another writer, Tom, made. We need to recover Tom's version of the article then reapply my changes. Tom's change was in PR#2437. The file in question is azure-resource-manager/powershell-azure-resource-manager.md	
Commands and actions	**What happens and why**
1. Go to https://github.com/Microsoft/azure-docs-pr/pull/2437/files and scroll down to the file. 2. Click the View button on the title bar of the diff display of that file. This takes to you the updated version of the file in GitHub. 3. Click the History button on the header bar of the file viewer pane. This shows you the commit history for that file. 4. Click on commit history for the version of the file you want. On the right side of the page you will see the full SHA for this commit. 5. Copy the SHA value for the commit. In this case the SHA is 30218c2013292a951253757bba9cef1beae3d7ae	The secret to this is that you need to find the SHA associated with the version of the file you want restored.
6. Check out the branch that overwrote the file. `git checkout mybranch`	This can be the branch where you did the bulk update. Or this could be a new branch for restore this one file.

7. Restore the previous version with the following command:
 	
`git checkout <SHA of commit> -- <path to file>`
 	
For example:
 	
`git checkout 30218c2013292a951253757bba9cef1beae3d7ae -- azure-resource-manager/powershell-azure-resource-manager.md`
	Now the file has been restored.
	
	You can use this same checkout process to restore any number of files. This could be useful if you accidentally deleted files and need them restored.

8. Update the file as necessary.	Use your favorite editor.

9. Add and commit your changes.
 	
`git add --all`
`git commit -m "description of the changes"`	

10. Push your changes up into your fork. `git push origin mybranch`	Now your fork is in sync with your local repository. You are ready to send a pull request to have your changes merged into the official repository.

Git task title

Commands and actions	**What happens and why**
1.	
2.	
3.	
4.	
Notes	