Understanding Git and GitHub

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](https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols) of the *Pro Git* book*.* There are other hosted git services available on the internet (BitBucket, Codeplex, etc.).

GitHub hierarchy

* Organization/Account (example: Microsoft or Azure)
  + Repository (example: WindowsServerDocs-pr or azure-content-pr)
    - Branch (example: master)
  + Fork – a clone of a repository
    - Branch (example: master)
    - Branch (example: July2016Freshness)

Forks and why you need one

* A Fork is a clone of a repository hosted on a Git server 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 are not running a git service, therefore Github cannot pull the clone on your local machine. You mush push your changed into your remote fork on GitHub.
* You cannot write (push) to the Microsoft repositories. You must send a Pull Request. Then the admins of the Microsoft 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.

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.



Git Object Model

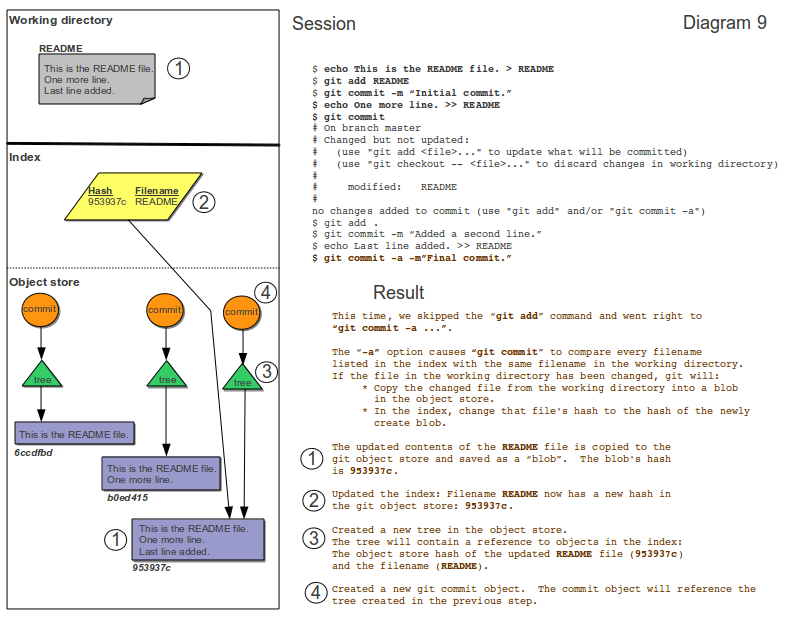
A git repository is defined by the data stored in the hidden .git folder on the local file system in the root folder of the repository. Git tracks the state of the repository in a database called ‘index’ and collection of files and folders known as the git object store.

### Git Object Types

|  |  |
| --- | --- |
| Blob object  http://www.gitguys.com/gitguys/db/images/blobonly.png | 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  http://www.gitguys.com/gitguys/db/images/treeonly.png | 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  http://www.gitguys.com/gitguys/db/images/commitonly.png | 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  http://www.gitguys.com/gitguys/db/images/tag.png | 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.



For a complete, step by step description of this process please refer to the source article from GitGuys at: <http://www.gitguys.com/topics/whats-the-deal-with-the-git-index/>

Setting up your working environment

Follow the instructions for setting up the tools as described in the Azure Contributor Guide [Tools and Setup](https://github.com/Azure/azure-content-pr/blob/master/contributor-guide/tools-and-setup.md) document for the following tasks:

* Creating a GitHub account and setting up your profile
* Creating Disqus account
* Configuring permissions in GitHub and 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 GitHub Desktop client for Windows

The GitHub Desktop client for Windows includes the following components:

* A very nice, modern Windows GUI application for managing git repositories
* A full install of the standard Git client for Windows (i.e. Git GUI and Git Bash)
* [Posh-Git](https://github.com/dahlbyk/posh-git) – a PowerShell environment for using the git client instead of Bash or CMD

The best reason for this using this client is the fact that you get a PowerShell command environment rather than Bash or CMD. This will be easier to support and customize and allows you to run the build scripts without having to launch a different shell environment.

### Installation steps

1. Download and run the GitHub Desktop client from <https://desktop.github.com/>
2. Open a new instance Git Shell and use git commands as described in the Contributor Guide

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](https://go.microsoft.com/fwlink/?LinkID=534107) 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.

|  |
| --- |
| **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:** Auto-Open Markdown Preview  **Install command:** ext install vscode-auto-open-markdown-preview  **Description:** This VS Code extension automatically shows Markdown preview whenever you open new Markdown file. |
| **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:** Spelling and Grammar Checker  **Install command:** ext install Spell  **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:** GitHub Clean White Theme  **Install command:** ext install Theme-GitHubCleanWhite  **Description:** This adds a theme to the VS Code theme library that changes the UI color coding and style formatting to match GitHub as closely as possible. That way your user experience will closely match what you seen in the GitHub web interface. |

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-content-pr or WindowServerDocs-pr).  Community contributors need to fork the public repo. |
| 1. Clone your fork to your local machine   Open your Git Shell and run the following commands:  cd C:\github  git clone https://[your GitHub user name]:[token]@github.com/<your GitHub user name>/azure-content-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. |
| 1. Create the upstream reference to the official source repository.   Run the following command from your Git Shell:  cd C:\github\azure-content-pr  git remote add upstream https://[your GitHub user name]:[token]@github.com/Azure/azure-content-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-content-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. |
| 1. 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. |
| 1. Make additions and changes to your content. | This is done using your content editing and creation tools like VS Code, Atom, TextMate, etc. |
| 1. 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. |
| 1. Commit your changes.   git commit -m "describe of the changes" | This checks-in the changes to your local git repository. |
| 1. 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. |
| 1. 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. |
| 1. 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** | |
|  | |

### 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-content-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). |
| 1. 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. |
| 1. 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. |
| 1. 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. | |

### Throw away an uncommitted branch and start over

|  |  |
| --- | --- |
| **Commands and actions** | **What happens and why** |
| 1. Git reset --hard |  |
|  |  |
|  |  |
|  |  |
| **Notes** | |
|  | |

### Git task title

|  |  |
| --- | --- |
| **Commands and actions** | **What happens and why** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Notes** | |
|  | |

### Git task title

|  |  |
| --- | --- |
| **Commands and actions** | **What happens and why** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Notes** | |
|  | |

### Git task title

|  |  |
| --- | --- |
| **Commands and actions** | **What happens and why** |
|  |  |
|  |  |
|  |  |
|  |  |
| **Notes** | |
|  | |