# GIT Revision Control

The following notes pertain to GIT and GIThub when doing revision control.

When one or more files has changed

**Using Git Bash**

* To display which files have changed

$ git status provides detail

$ git status –s provides an overview of status

* To examine how a file has changed

$ git diff *path\_of\_file*

* To stage a changed file (or files)

$ git add *path\_of\_file* (‘\*’ denoting wild-card is permitted)

* To stage all of the changed files

$ git add –u

Git doesn’t provide any indication that the files have been staged, but you can use “$ git status” or “$ git status –s” to see the change of status.

* To examine changes for all of the staged files

$ git diff - -staged

$ git commit [*options and parameters*] prompts you for a description of the changes that are being committed. It does this by popping up a window in the editor program that you specified during the personalization of the git install process.

* To commit all staged files

$ git commit

* To **add** and **commit** all changed files (but ignore new files)

$ git commit -a

* To display history use a variation of $ git log . . .

$ git log provides a low-detail overview

$ git log –p provides details

$ git log –p -2 provides details for the most recent 2 **commit**s

$ git log –p –since=2.days provides details for **commit**s that occurred in last 2 days

(you could also say “3.days” or “2.weeks”)

$ git log –p –since=2020-08-05 provides details for **commit**s that occurred after 8/2/2020

$ git log - -name-only to show commits with paths of the committed files

* To revise the description that was just supplied for a **Commit**

$ git commit –amend

Git pops up the editor program which contains your descriptive text; modify it, save it, and close the editor.

* To revise a **commit**, because you forgot to modify or stage another file (or maybe more than one file), make the file modifications (if not already made), stage the files, and then

$ git commit -amend

Git pops up the editor program which contains your descriptive text; modify it, save it, and close the editor.

* To undo file editing, and to restore the file to the contents that it had as of the last **commit**

$ git restore *path\_of\_file*

*path\_of\_file* may contain the wildcard designator (\*) to indicate more than one file

* To copy recent git activity to GitHub – for the current repository

$ git push

You will be asked to supply the SSH passphrase. This works only if there are no other contributors. A more elaborate set of commands will be needed (to reconcile) if work is being done on this repository concurrently by other team members.

**Using Git GUI**

* Start GitGui; git responds with a dialog titled Git GUI.
* You have 2 options to select the repository: Click **Open Existing Repository** to use a folder browser, or click the name of the repository if it is listed under **Open Recent Repository:** (The pull-down menu **Repository** – on the menu bar – provides the same 2 options in a different format.)
* The upper-left corner of the dialog - under **Unstaged Changes** - shows the names of the files that have been changed. It also shows files that have not yet been added (unless these files are listed in .gitignore). (I have noticed that Git GUI is sometimes not aware of not-yet-added files in a subfolder; this misbehavior stops when the subfolder becomes known to git, e.g. as a consequence of adding at least one file in that subfolder via git Bash.)
* Click the “name” part of one of the files under **Unstaged Changes**; Git GUI responds by displaying a brief indication of the changes in the upper-right panel and it displays **Untracked, not staged** in the title of this panel.
* To stage a changed file, click the “file icon” part of the file; Git GUI responds by moving the file from the list under **Unstaged Changes** to the list under **Staged Changes (Will Commit)**.
* Click the “name” part of one of the files under **Staged Changes (Will Commit)**; Git GUI responds by displaying a brief indication of the changes in the upper-right panel and it displays **Staged for Commit** in the title of this panel.
* You can change the status of a file from “staged” to “unstaged” by clicking the “file icon” part of the file; Git GUI responds by moving the file from the list under **Staged Changes (Will Commit)** back to the list under **Unstaged Changes**.
* To commit the staged files, (1) type a description of the changes into the text box in the lower-left corner – under **Commit Message**, and (2) click the **Commit** command button.
* To revise the description that was just supplied for a **Commit**, click the **Amend Last Command** check box at the right edge of the dialog, revise the descriptive text, and click the **Commit** command button again.
* To copy recent git activity to GitHub – for the current repository, click the **Push** command button, and follow displayed instructions. You will be asked to supply the SSH passphrase.

Using GitHub

As a user of GitHub with very limited experience, I find it to be a convenient place to store my code and documents – with their revision history, i.e. a place where I can back up my git repositories.

When one signs into GitHub . . .

* Note the list of recent repositories in the upper-left corner of the landing page.
* If you are not on the landing page, you can navigate to your repositories by clicking the down-arrow to the right of the avatar in the upper-right corner. GitHub displays a pull-down menu. Click **Your repositories**.

You can drill down starting from the repository list.

* Start by clicking the name of a repository. GitHub responds by displaying the repository contents – files and folders.
* You can click the name of a folder to view its contents.
* You can click the name of a file to see its text.
* GitHub also displays describes the latest activity (commit plus date/time). Toward the right of this description is a hyperlink titled **History**.
* Click **History** to request a history of revisions.

One can look at a log of all (perhaps recent) GitHub activity that you have performed.

* Click the down-arrow to the right of the avatar in the upper-right corner. GitHub displays a pull-down menu.
* Click **Settings** in the pull-down menu. GitHub displays a page titled **Public Profile**. Toward the left there is a menu of options headed by *user\_ID* and “Personal settings”.
* Click **Security log**. GitHub displays the activity in reverse chronological order.

USING CLONE

Perhaps the most important reason for a remote repository – like GitHub – is that provides the ability to collaborate with users on other computers. All of the team players must set up GitHub accounts. If one particular user does not have a local copy of a Git repository on his computer, he can use the CLONEcommand as an easy way to put a copy of the repository on his computer. The steps are as follows.

* Navigate to GitHub. When GitHub starts, by default it displays the names of the repositories in the left-hand pane.
* Right-click the name of the repository that you want to clone. GitHub displays a pop-up menu.
* Click **Copy link address** in this menu. This puts the HTTPS URL of the selected repository into the clipboard.
* As a precaution against accidentally erasing this from the clipboard, it might be advisable to copy this URL into a notepad file on the local computer. There is no need to keep GitHub open, and one can sign out.
* Start GitBash, and navigate to an empty folder on the local computer, where you want the local repository to reside.
* Type **git clone**; then type the <insert> key to paste the URL into the GitBash command. Type the <Enter> key. Git writes a copy of the repository into the designated folder.

Visual Studio

Using git has an effect on Visual Studio.

* The Team Explorer page will show files that are not yet tracked or that have been modified.
* You will be able to find (under **Actions** drop-down menu a command button to view history.
* You can right-click a source file that is being tracked by git, and Visual Studio provides the option to view that file.
* From the revision history of a source file, you can select two of them to view differences. This view is easier to read than the differences shown via Git Bash or Git Gui.

Much of Git and GitHub can be conducted from Visual Studio. Refer to the Pluralsight course “Using Git For Source Control In Visual Studio 2019”. The examples presented in this course mostly use an alternative to GitHub, AzureDevOps, which (like GitHub) provides a free remote repository. The instructor suggests <https://visualstudio.github.com> to learn details and to download Visual Studio extensions for GitHub.

On my 1st attempt to use AzureDevOps as a remote repository, I made a mess. My 1st mistake was referring to the wrong organization (sncole0974) from the newer (Windows 10) computer; I should have chosen sncole00 for the organization. Subsequent attempts to resolve the problems possibly contributed to the mess. Ultimately I needed some way to reverse my steps and to undo the connection between that Visual Studio project and Azure DevOps. The following steps worked.

* Use Git Bash, and navigate to the folder that contains the Visual Studio project.
* Type **$ git remote** to get a list of the connections to a remote repository. The response that I saw was **origin**. This indicates that “origin” is the name of the remote; therefore, I used “origin” in the last argument of the next command.
* **$ git remote rm origin** The result of this command was - - when I went back to

Visual Studio -> Team Explorer -> Home -> Sync

Visual Studio again displayed the invitation

**Backup and share your code. Publish it to a Git service.**

References:

1. “Using Git for Source Control In Visual Studio 2010”. GitRepositories\WebAppCourseNotes\UsingGitForSourceControlInVisualStudio2019\UsingGit4SrcCtlInVS2019.docx, section “Push to a Remote Repository”.
2. The internet web page “How to Remove a Git Remote”. (Use that title for keywords.)