# Title Page

**Git and GitHub**

|  |  |
| --- | --- |
| Image result for git | Image result for github |

Figure 1: Logo for Git and GitHub

## Publication Number

AFSL-2018-04

See \\AFSL\TechnicalDataPackage\AFSLPublicationNumbers.docx for list of publication numbers.

## Date of Issue

April 28, 2018

# Record of Manual Revisions

Table 1: Record of manual revisions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Revision** | **Date** | **Pages Affected** | **Revisions** | **Author** | **Check** | **Approved** |
| 1 | 03/18/18 | All | Created document | Christopher Lum |  | Christopher Lum |
| 2 | 03/23/18 | Various | Updated notes and cleaned up document | Christopher Lum |  | Christopher Lum |
| 3 | 04/11/18 | 8 | Added command to rollback to previous commit | Christopher Lum |  | Christopher Lum |
| 4 | 04/23/18 | Various | Discuss posh-git install | Christopher Lum |  | Christopher Lum |
| 5 | 04/28/18 | 6 | Added notes on PowerShell script install | Christopher Lum |  | Christopher Lum |
| 6 | 07/02/18 | 13 | Added section on GitHub |  |  |  |

# Table of Contents

Contents

[1. Title Page 1](#_Toc509114289)

[1. Publication Number 1](#_Toc509114290)

[2. Date of Issue 1](#_Toc509114291)

[2. Record of Manual Revisions 1](#_Toc509114292)

[3. Table of Contents 2](#_Toc509114293)

[4. Nomenclature & Glossary 3](#_Toc509114294)

[5. Introduction 4](#_Toc509114295)

[1. Contact Info 4](#_Toc509114296)

[2. Other Documentation 4](#_Toc509114297)

[6. Git Command Line 5](#_Toc509114298)

[1. Typical Workflow Commands 5](#_Toc509114299)

[2. Helpful Tools/Commands 6](#_Toc509114300)

[7. GitHub Desktop App 7](#_Toc509114301)

[1. Setup 7](#_Toc509114302)

[8. GitHub 8](#_Toc509114303)

[9. Bibliography 9](#_Toc509114304)

# Nomenclature & Glossary

Table 2: Nomenclature and glossary of terms

|  |  |  |
| --- | --- | --- |
| **Term** | **Definition** | **Comment** |
| AFSL | Autonomous Flight Systems Laboratory |  |
| Git |  | The general decentralized version control system |
| GitHub | Cloud service | A private company that hosts git repositories online |
| Git Shell |  | Windows application that provides a git command line interface |
| GitHub Desktop |  | Windows application that provides a git GUI interface |
| local | Local repository | Repo on local machine |
| remote | Remote repository | Repo on cloud service |
| SHA or SHA-1 | Secure Hash Algorithm 1 | SHA-1 is the algorithm used to generate the commit ID |
| UW | University of Washington |  |

# Introduction

This document contains notes related to the operation of Git and GitHub.

## Contact Info

Contact information for the maintainer of this document is:

Autonomous Flight Systems Laboratory

University of Washington

AERB Room 139, Box 352250

Seattle, WA

(206) 543-0539

lum@uw.edu

## Other Documentation

* YouTube tutorial: Git Tutorial for Beginners: Command-Line Fundamentals ([URL](https://www.youtube.com/watch?v=HVsySz-h9r4))

# Workstation Setup

There are several tools you will need to install to work with Git and GitHub.

## Git

We first download a distribution of Git onto the computer and modify the PATH so tools like the command line, PowerShell, etc. can use Git.

1. Download and install git for windows at <https://gitforwindows.org/> .
   1. Choose the default installation options.
   2. This will modify the PATH so other applications (for example PowerShell) can use Git.
2. Open PowerShell and configure some git settings

git config --global user.email “lum@uw.edu”

git config --global user.name “Christopher Lum”

## Posh-Git

We now add the posh-git package to PowerShell.

1. Open PowerShell.
2. Find the GitHub repository for Posh-Git at <https://github.com/dahlbyk/posh-git> and clone this to your machine (see section 7.1).
3. Navigate inside the ‘posh-git’ repository.
4. Execute the install script[[1]](#footnote-2)

.\install.ps1

1. You should now see a turquoise [master =] in the PowerShell prompt (Figure 2).

\\LUMTHINKPADTAB2\Users\Alison\Desktop\TEMP\Capture02.PNG

Figure 2: posh-git running on PowerShell.

## GitHub Desktop App

1. Download and install the GitHub Desktop application from [http://desktop.github.com](http://desktop.github.com/)

# Git Command Line

This section discusses interacting with git via the command line. Git Shell or PowerShell with posh-git are two useful application to use the command line.

## Cloning a Repository

The first thing to do is to clone a repository. These instructions are based on a YouTube video entitled “Git Clone with Terminal” located [here](https://www.youtube.com/watch?v=3wU8a80GMrU).

1. Find the URL of your repository.
2. Start Git Shell or PowerShell.
3. Change working directory to the location where you would like to download/clone the repository.
4. Clone the repository using ‘git clone ***<URL>***’. For example

git clone https://github.com/clum/LearningGit.git

1. Change working directory to the newly downloaded/cloned directory.

You may want to add this to repository to the GitHub Desktop app so that it shows up in the app. This is discussed in section 8.8.

## Typical Workflow Commands

A typical workflow in Git as follows:

1. Pull changes from the remote to your local repo.
2. Create a new branch to develop a new feature.
3. Do some development in this new branch locally.
4. Push the new branch to the remote repo.
5. Merge the new branch into the master branch.
6. Delete the new feature branch on both the local and remote repo.

A typical session is shown in Table 3. Extra comments on helpful commands during a typical session are shown in .

Table 3: Git commands for a typical session.

|  |  |
| --- | --- |
| git pull origin master | //pull latest changes |
|  |  |
| //create a new branch |  |
| git branch NewFeatureBranch | //create NewFeatureBranch on local repo |
| git checkout NewFeatureBranch | //switch to NewFeatureBranch |
|  |  |
| //modify or add files |  |
| git add -A | //stage modified/new files |
| git commit -m “Added new feature” | //commit staged files locally |
|  |  |
| //push to remote |  |
| git pull origin NewFeatureBranch | //pull before pushing |
| git push origin NewFeatureBranch | //push the new branch to the remote |
|  |  |
| //merge NewFeatureBranch into master |  |
| git checkout master | //switch to master |
| git merge NewFeatureBranch | //merge NewFeatureBranch into master |
| git pull origin master | //pull before pushing |
| git push origin master | //push newly merged master to the remote repo |
|  |  |
| //cleanup/delete branches |  |
| git branch –d NewFeatureBranch | //delete NewFeatureBranch from local repo |
| git push origin --delete NewFeatureBranch | //delete NewFeatureBranch from remote repo |

Table 4: Commands used during a typical Git session

|  |  |  |
| --- | --- | --- |
| **Command** | **Result** | **Comment/Example** |
|  |  |  |
| git add ***<PathFileName>*** | Add a modified/new file to the staging area | “git add –A” to stage all files. |
| git reset ***<PathFileName>*** | Remove a file form the staging area | “git reset” to unstage all files |
| git checkout -- . | Revert all unstaged files | This only reverts modifications to existing files, it does not remove new untracked files. |
| git clean -f | Clear/delete untracked files | This can be helpful to remove added files that have not been staged. Use –n instead of –f to see which files will be removed. |
| git commit –m “***<message>***” | Commit staged files w/ message | git commit –m “Revising files” |
| git pull ***<RemoteAlias> <BranchName>*** | Pull from remote |  |
| git push ***<RemoteAlias> <BranchName>*** | Push committed changes to the branch on the specified remote | git push origin master |
| git push origin --delete ***<BranchName>*** | Delete a branch from the remote repository | See [URL](https://www.youtube.com/watch?v=HVsySz-h9r4) @ 27:25 |
| git branch ***<BranchName>*** | Create the named branch | git branch NewFeatureBranch |
| git branch -a | See all branches |  |
| git checkout ***<BranchName>*** | Switch to the named branch |  |
| git merge ***<BranchName>*** | Merge BranchName into the current branch | Be sure the push after merging |
| git branch –d ***<BranchName>*** | Delete named branch from the local repository | Be sure to push after merging |

## Helpful Tools/Commands

Some helpful commands and tools that may or may not be part of a standard git workflow is shown in Table 5.

Table 5: Helpful tools and commands in Git command line

|  |  |  |
| --- | --- | --- |
| **Command** | **Result** | **Comment/Example** |
| git config --global user.name “***<name>***” | Set the name of the person who will be making commits | git config --global user.name “Christopher Lum” |
| git config --global user.email “***<email>***” | Set the email name of the person who will be making commits | git config --global user.email “lum@uw.edu” |
| git log | Get a log of commits |  |
| git remote –v | Show remote alias (-v shows the URL to the repository) |  |
| gitk ***<PathFileName>*** | Version history of a file | [URL](https://stackoverflow.com/questions/9807393/git-show-history-of-a-file/9807432), see Figure 3. |
| git config --list | List all configurations |  |
| git help <verb> | Get help | git help config |
| git diff | Show differences in staged files |  |
| git difftool | Diff tool |  |
| git checkout -b ***<BranchName> <SHA>*** | Rollback to a previous commit and create a new branch to hold subsequent changes | [URL 1](https://git-scm.com/docs/git-checkout" \l "_detached_head)  [URL](https://stackoverflow.com/questions/4114095/how-to-revert-git-repository-to-a-previous-commit) 2  [URL](https://www.git-tower.com/learn/git/faq/detached-head-when-checkout-commit) 3 |

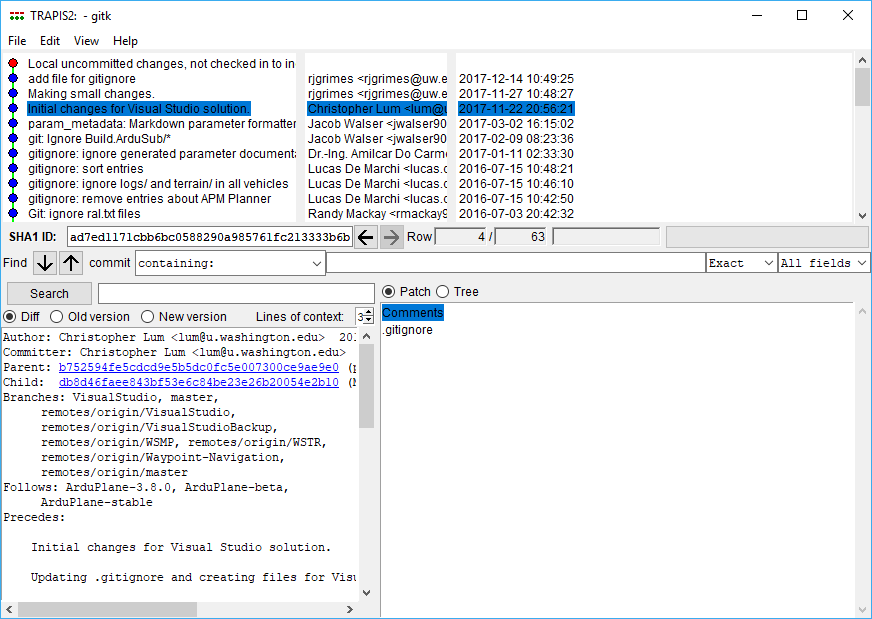


Figure 3: Inspecting version history of a file.

# GitHub Desktop App

## Setup

You should ensure that the GitHub Desktop application has the appropriate settings.

1. Click on the settings icon (upper right) > Options…
2. Accounts: Ensure you are logged in as yourself.
3. Configure git: This should be consistent with the account you logged in with otherwise, there may be inconsistencies during commits.
4. Clone path: Set this to the location where you would like clone repositories to be downloaded onto your local machine.

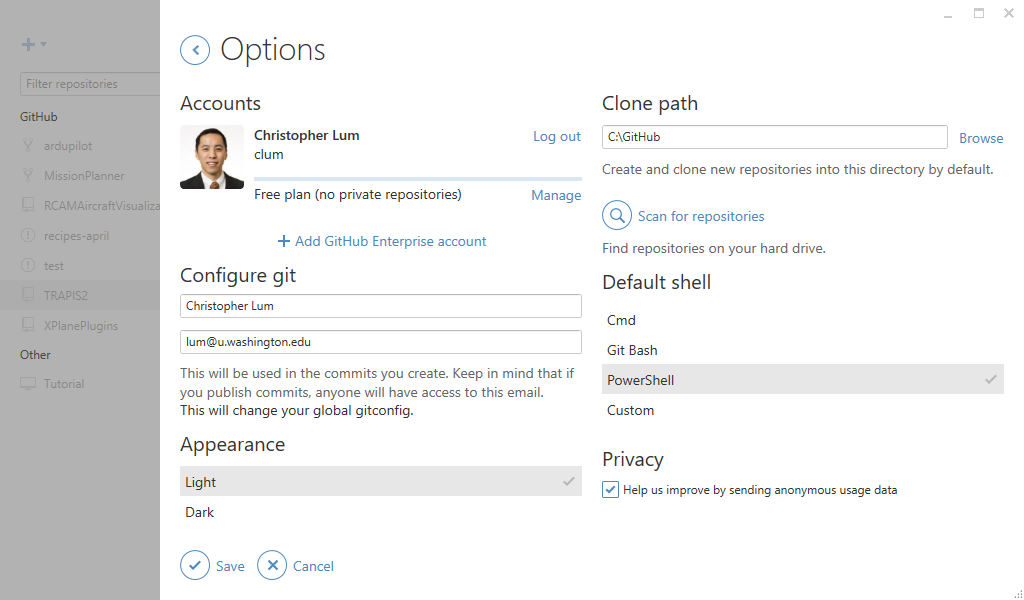


Figure 4: Settings for GitHub Desktop app.

## Cloning a Repository

You can use the GitHub Desktop to clone a repository or do this manually using methods discussed above.

## Adding an Existing Repository

If you have cloned a repository using Git command line, you may need to add it to the GitHub Desktop app.

1. Click on repositories icon (+ sign in the upper left) > select ‘Add’ > browse to the location of the local repo > click ‘Add repository’ (see Figure 5).
2. This repo should how appear in the list of repos in GitHub Desktop app (Figure 6).

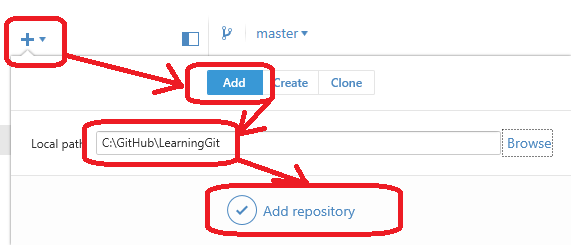


Figure 5: Adding a local repo to the GitHub Desktop app.

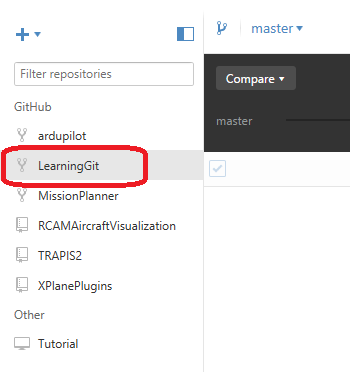


Figure 6: Repo now appears in the GitHub Desktop app.

# GitHub

GitHub is the private cloud resource that is used to host Git repositories.

## Joining the AFSL GitHub Organization

The official UW AFSL GitHub organization can be found [here](https://github.com/uwafsl). To become a member, contact an owner of the organization and request an invitation. To find an owner, navigate to the [People tab](https://github.com/orgs/uwafsl/people) to see who is listed as an “Owner”, or simply ask around the lab.

## Git/GitHub Guidelines

As a contributor to any of UW AFSL’s GitHub projects, you are expect to abide by several guidelines to ensure software development goes smoothly for everyone:

* **Never push directly to the master branch**. Instead, create a fork or a separate branch to harbor code changes and [submit a pull request](https://help.github.com/articles/creating-a-pull-request/). Have another contributor review this pull request, look for messy code, and ensure your code builds before merging it with master. The exception to this rule is if you are the only one working on a project and there is hence no one available to process your pull request. But you should still develop changes on a separate branch and confirm they work before merging with master.
* **Write clear and concise commit messages**. View these [seven rules](https://chris.beams.io/posts/git-commit/" \l "seven-rules).
* **Limit yourself to one feature/modification per commit**. Try not to include two or more unrelated changes in a single commit. If you do, make sure it is clear in the commit message. To be clear, it’s fine to edit multiple lines and files in a single commit. Just make sure these commits work toward a single goal, such as “Implement X feature”, “Fix typos in comments”, “Add unit tests for X”, etc.

# Bibliography

**There are no sources in the current documen**

1. If you are unable to execute this script, you may need to set the execution policy to unrestricted. To do this first run PowerShell as an administrator then type in ‘Set-ExecutionPolicy unrestricted’ [↑](#footnote-ref-2)