Using Git for OpenVSP Model Version Control

2020 OpenVSP Workshop, Virtual Broadcast

September 15-17, 2020

BRANDON LITHERLAND, AST
NASA LANGLEY RESEARCH CENTER
AERONAUTICS SYSTEMS ANALYSIS BRANCH

Outline

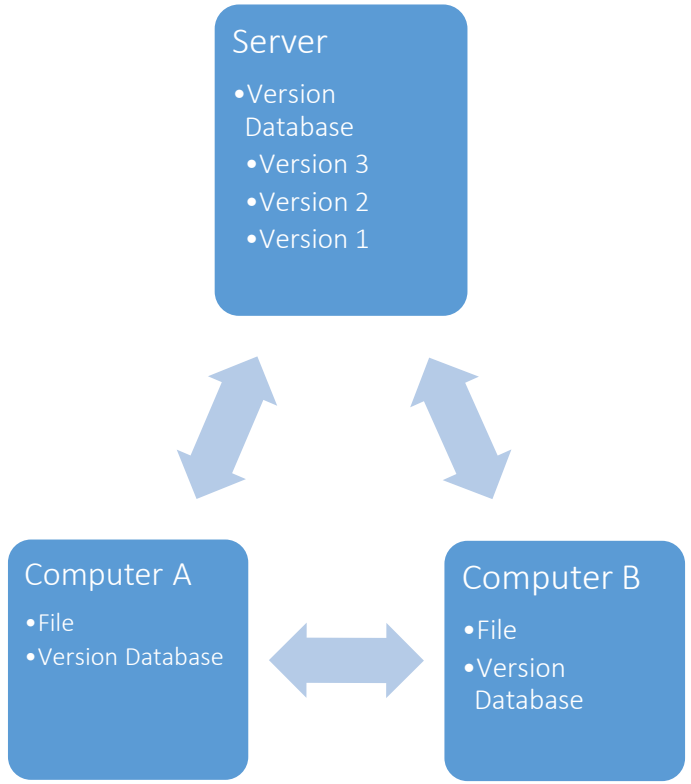
- Why use Git?
- Nomenclature
- Git-Bash
  - Basic commands
- GitHub & GitLab
- GitHub Desktop
- Demo (time permitting)

Context

- We are discussing version control for models rather than for software source code.
  - This can include geometry, performance models, design inputs, notes, scripts, etc…
- Applies to single/multiple designers working on a model.
  - Each individual can modify the models as needed at the same time and merge changes when complete.
  - Enables extensive collaboration.
- Avoids saving multiple copies of the same model as the design progresses (“backups”).

Why Use Git?

- Simple, clean, easy to use version control of various file types.
- You can save files as much as you want without affecting the stored version.
  - Your changes do not affect the controlled model until you “commit” the changes.
- You can revert to any previous version at any time without losing the most recent work.
- Highly collaborative environment on user-controlled resource.
- Can work locally on your laptop or online at GitHub or GitLab. Changes are tracked on both.
- You aren’t just working with a local copy, you work with the entire history of the project.
  - Distributed Version Control System (DVCS)
  - There are multiple backups of the project in case something is corrupted or lost.
- OpenVSP source code is on GitHub!
  - https://github.com/OpenVSP/OpenVSP



Why Use Git?

- VSP models are XML-based code
- Few VSP files are unformatted.
- Traditionally used for software.
  - Works well with human-readable formats.
- You can version control *everything* in the repository:
  - VSP scripts
  - Python code/scripts
  - Design files
  - Variables
  - Configurations
  - Analysis input & output files
  - PDF, images, Excel, etc.
    - Unformatted or binary files are still tracked but you cannot see the differences. Reverting can be difficult.
- Documentation uses MarkDown language. Useful for keeping a summary or description.

```
<?xml version="1.0"?>
<Vsp_Geometry>
  <Version>4</Version>
  <Vehicle>
    <ParmContainer>
      <ID>XGWDMTNJCK</ID>
      <Name>Cessna_337_Skymaster</Name>
      <BBox>
        <X_Len Value="2.95380220820890710e+001" ID="RLRZFCKBUNB"/>
        <X_Min Value="-1.30000000000000000e+000" ID="JBYJZXSHDTA"/>
        <Y_Len Value="3.79593691524678790e+001" ID="WMFTWSAJFBP"/>
        <Y_Min Value="-1.89796845762339390e+001" ID="DVBDNNYCSSZ"/>
        <Z_Len Value="8.22848148042333350e+000" ID="NESDHAOCLBY"/>
        <Z_Min Value="-3.11038148042333340e+000" ID="PDGATXWMJND"/>
      </BBox>
      <Design>
        <Working_XDDM_Type Value="0.00000000000000000e+000" ID="LNBCWPTXDRD"/>
      </Design>
      <ExportFlag>
        <CompGeom_CSV_Export Value="1.00000000000000000e+000" ID="ZMUKXNUYOFS"/>
        <DegenGeom_CSV_Export Value="1.00000000000000000e+000" ID="JPICRTUYSIX"/>
        <DegenGeom_M_Export Value="1.00000000000000000e+000" ID="GVYMNTADEII"/>
        <DragBuild_TSV_Export Value="1.00000000000000000e+000" ID="JCVTEMKMPJU"/>
      </ExportFlag>
      <FitModel>
        <Select_Box_Flag Value="0.00000000000000000e+000" ID="BIPTTOINLWB"/>
        <Select_One_Flag Value="0.00000000000000000e+000" ID="AVNHDNTVURD"/>
        <U_TargetPt Value="0.00000000000000000e+000" ID="JNQDQHDFQYA"/>
        <U_Type Value="1.00000000000000000e+000" ID="GFGLKORGTXI"/>
        <W_TargetPt Value="0.00000000000000000e+000" ID="IHZPNQMLERK"/>
        <W_Type Value="1.00000000000000000e+000" ID="UTKACGWGTQQ"/>
      </FitModel>
```

```
# DEGENERATE GEOMETRY CSV FILE

# NUMBER OF COMPONENTS
2

# DegenGeom Type, Name, SurfNdx, GeomID
LIFTING_SURFACE,WingGeom,0,JFCMTJETXJ
# DegenGeom Type,nXsecs, nPnts/Xsec
SURFACE_NODE,49,21
# x,y,z,u,w
1.00000000000000000e+00, 0.00000000000000000e+00,
9.83863513605328199e-01, 0.00000000000000000e+00,
9.46238966883103938e-01, 0.00000000000000000e+00,
8.64561603940949163e-01, 0.00000000000000000e+00,
7.11856938484336954e-01, 0.00000000000000000e+00,
4.92460161371229721e-01, 0.00000000000000000e+00,
2.72462665329947062e-01, 0.00000000000000000e+00,
1.19271905909347900e-01, 0.00000000000000000e+00,
3.90483431915742557e-02, 0.00000000000000000e+00,
6.18945895744581892e-03, 0.00000000000000000e+00,
```

```
Sref = 45.000000
Cref = 2.500000
Bref = 18.000000
X_cg = 0.000000
Y_cg = 0.000000
Z_cg = 0.000000
Mach = 0.233300
AoA = 6.000000
Beta = 0.000000
Vinf = 100.000000
Rho = 0.002377
ReCref = 10000000.000000
ClMax = -1.000000
MaxTurningAngle = -1.000000
Symmetry = Y
FarDist = -1.000000
NumWakeNodes = -1
WakeIters = 5
NumberOfRotors = 0
NumberOfControlGroups = 0
Preconditioner = Matrix
Vortex Lift = Y
LE Suction = N
Karman-Tsien Correction = Y
```

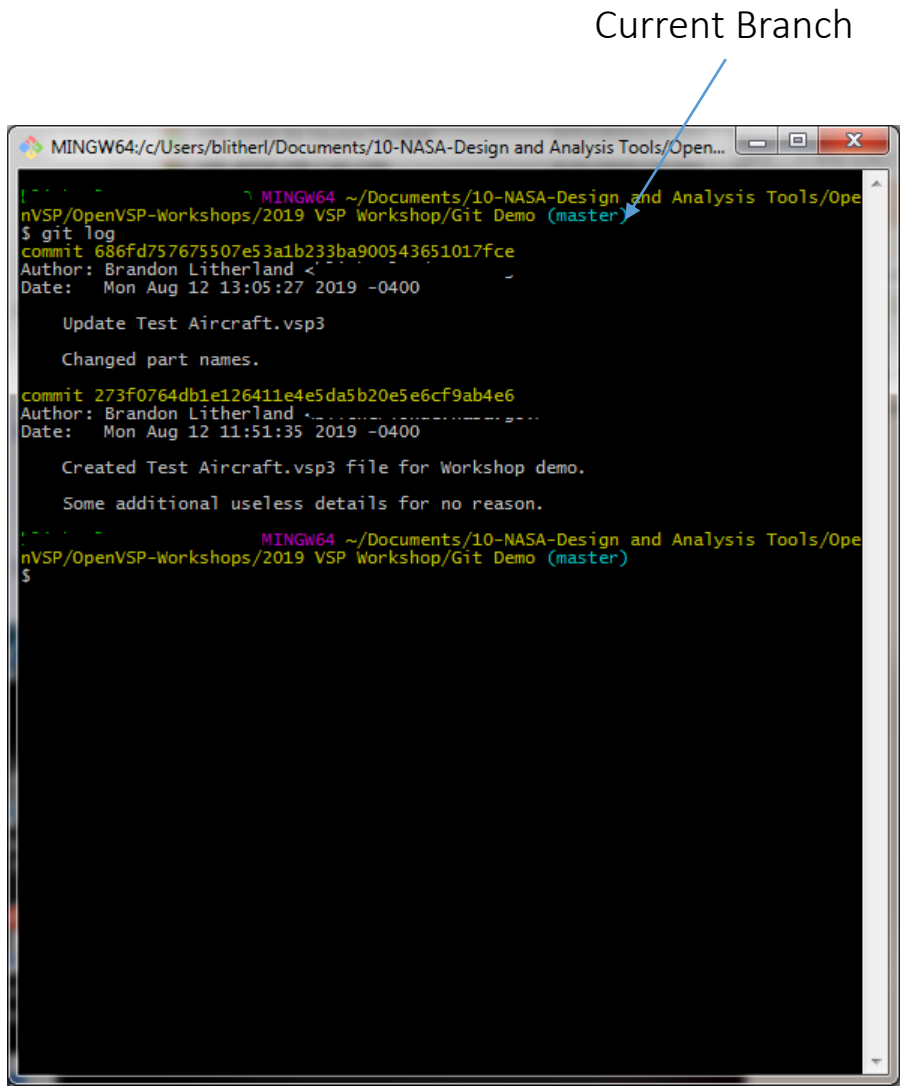
Git Nomenclature

- **Branches** – Parallel versions of the project where you can safely make changes, experiment, fix bugs, etc…
  - "I've checked out a version of the project to work on."
- **Fork** – GitHub (online only) term that implies making an online copy of the main repository of work to your user account.
  - "I'm working on MY copy of the project."
- **Clone** – Copy the online repository to my local storage.
- **Checkout** – Work with a specific project branch.
- **Commit** – You update the controlled version of the project with the saved copies stored locally.
  - "I'm happy with my saved changes and want those changes added to the project and history."
- **Push/Pull** – Either "push" changes from your local copy to the online repository or "pull" the information from one branch to another.
  - "I'm done making commits to the project for now and am ready to update the main branch. I push my local changes to the online repository"
  - "I request to pull the changes from my online branch to the main branch." – When collaborating.

Git-bash

Git-bash - For those that enjoy command line interfaces: https://git-scm.com/download/

- Linux-style command structure for controlling local repositories and making changes online.
- Easy to learn and usually quicker than using a GUI.
- Current working branch is highlighted in blue.
- Several operations in Git can not be done online at GitHub. Local Git installation must be used.
- There is also a Git GUI but not very intuitive/useful.
- Git Cheat Sheet: https://services.github.com/on-demand/downloads/github-git-cheat-sheet.pdf



Git-bash

Useful Commands

`$ git init [project-name]`
- Creates a new local repository with the specified name.
- "`git init`" makes the current directory the repository.
- Makes the repository in the current directory.

`$ git clone [url]`
- Downloads a project and its entire version history.
- Makes the repository in the current directory.

==========

`$ git branch`
- Lists all local branches in the current repository.

`$ git branch [branch-name]`
- Creates a new branch.

`$ git checkout [branch-name]`
- Switches to the specified branch and updates working directory.

`$ git merge [branch-name]`
- Combines the specified branch's history into the current branch.

`$ git branch -d [branch-name]`
- Deletes the specified branch.
- Good for removing branches that are no longer needed.

`$ git status`
- Lists all new or modified files to be committed.

`$ git diff`
- Shows file differences not yet staged.

`$ git add [file]`
- Snapshots the file in preparation for versioning.

`$ git diff --staged`
- Shows file differences between staging and the last file version.

`$ git reset [file]`
- Unstages the file, but preserves its contents.

`$ git commit –m "[descriptive message]"`
- Records file snapshots permanently in version history.

`$ git pull`
- Fetches the online repository to your local copy.

`$ git push [remote] [branch]`
- Pushes local changes to the remote repository
- Ex: "`git push origin master`"

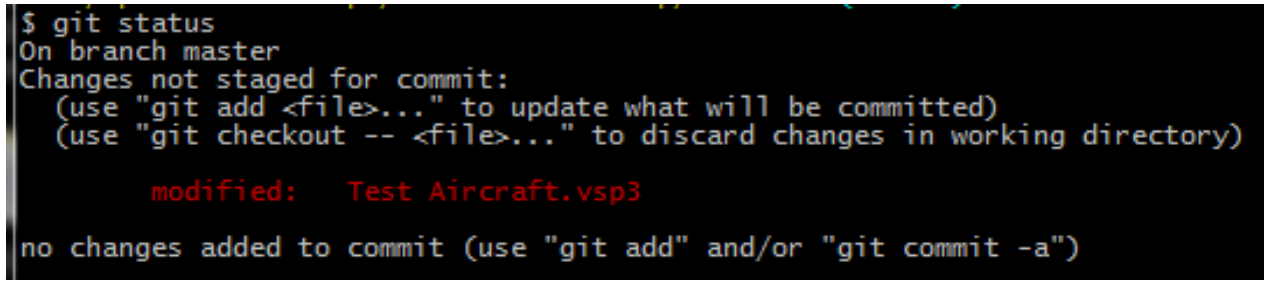
`$ git log`
- Shows commit history

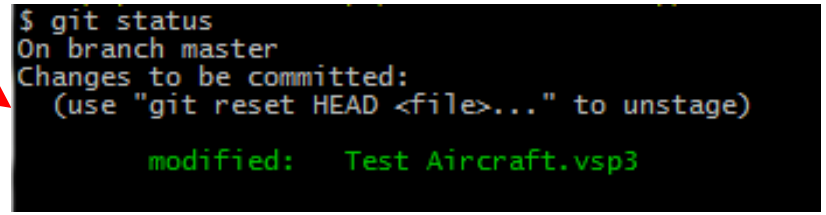
Git-bash

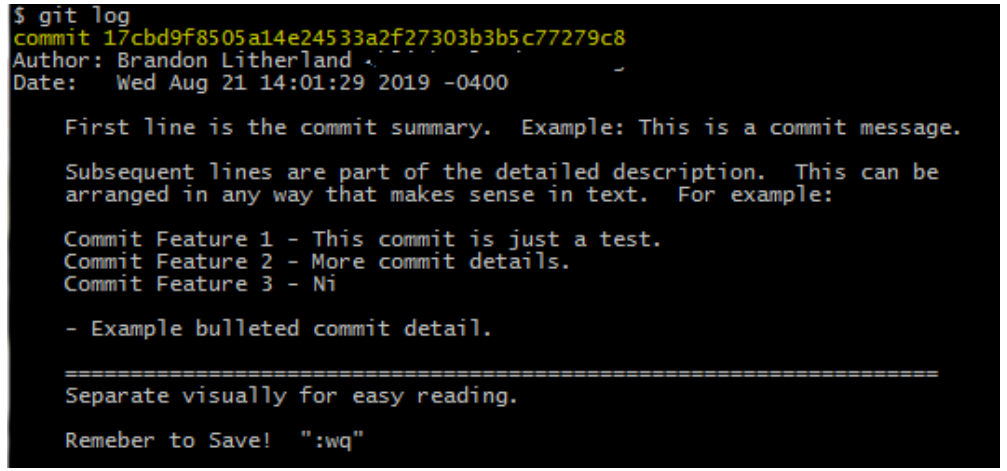
Example Workflow: Local Files

1. Navigate to desired directory.
2. `$ git init MyProject`
3. Add files to the repository.
    1. Create files using whatever software.
    2. Can have one or many files.
    3. Git is tracking changes as you save.
4. `$ git status`
    1. Uncommitted files are listed in RED.
5. `$ git add [filename]`
    1. Stages files for commit.
    2. Can add multiple files at once.
        1. Example: `$ git add *.vsp3` will add ALL VSP files at once.
        2. `$ git add .` will add ALL untracked or unstaged files!
    3. `$ git status` will now show staged files in GREEN.
6. `$ git commit –m “[comments]”`
    1. Changes are committed to the project and added to the history with [comments] describing the changes.

*Note: `$ git commit` will bring up the VIM editor where you can make a summary AND detailed description. The “Insert” key will let you edit the text. “Esc” will take you out of insert mode. To write the text and exit the editor, use “**:wq**” – Read as colon write quit. THIS IS MUCH BETTER PRACTICE FOR DESCRIBING COMMITS!!!







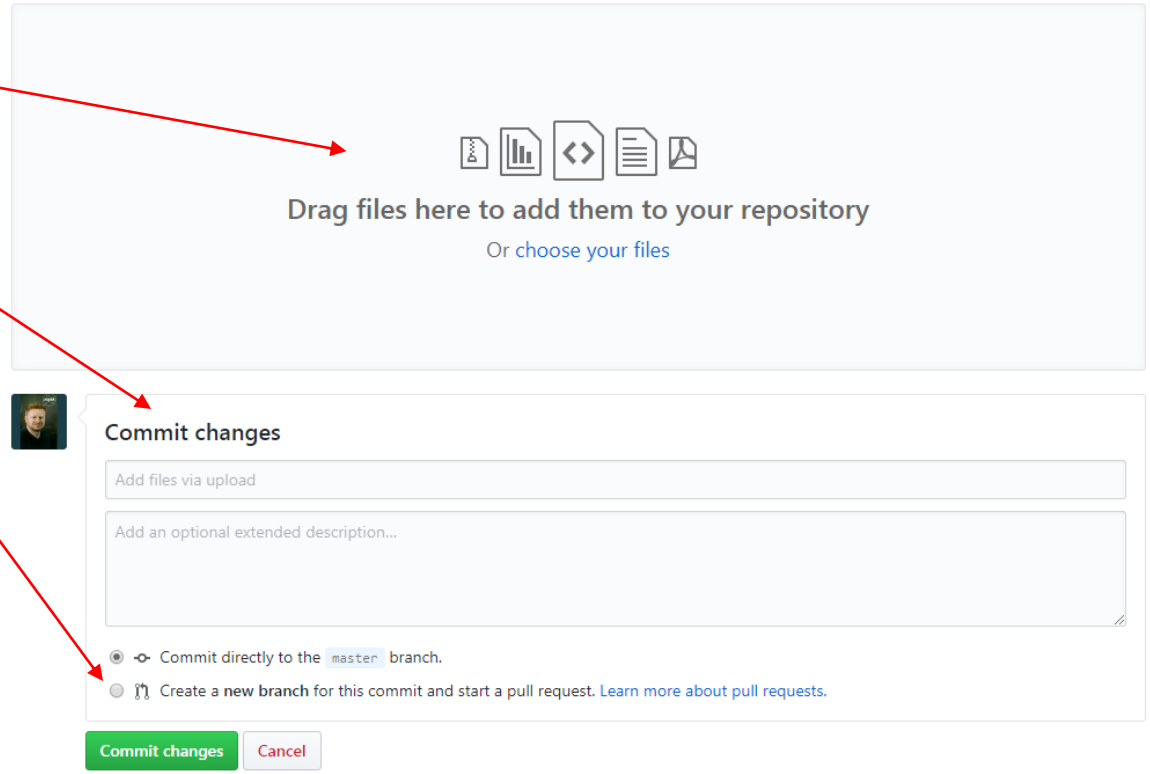
GitHub / GitLab

- This is where the online version control and collaboration happens (mostly).
- Similar to other systems, a multitude of helpful online tutorials are present and available.
  - Example: https://guides.github.com/activities/hello-world/
- Step-by-step instructions on navigating Git can be found, including operations in git-bash.
- Most operations here are simple point-and-click steps.
- Very helpful in examining the status of the project and seeing what versions are checked out.

GitHub / GitLab

GitHub (github.com): Online collaboration and backup version control

- Can add saved files
  - New versions of existing files or add new files.
- Perform Commits
- Initiate Pull Requests



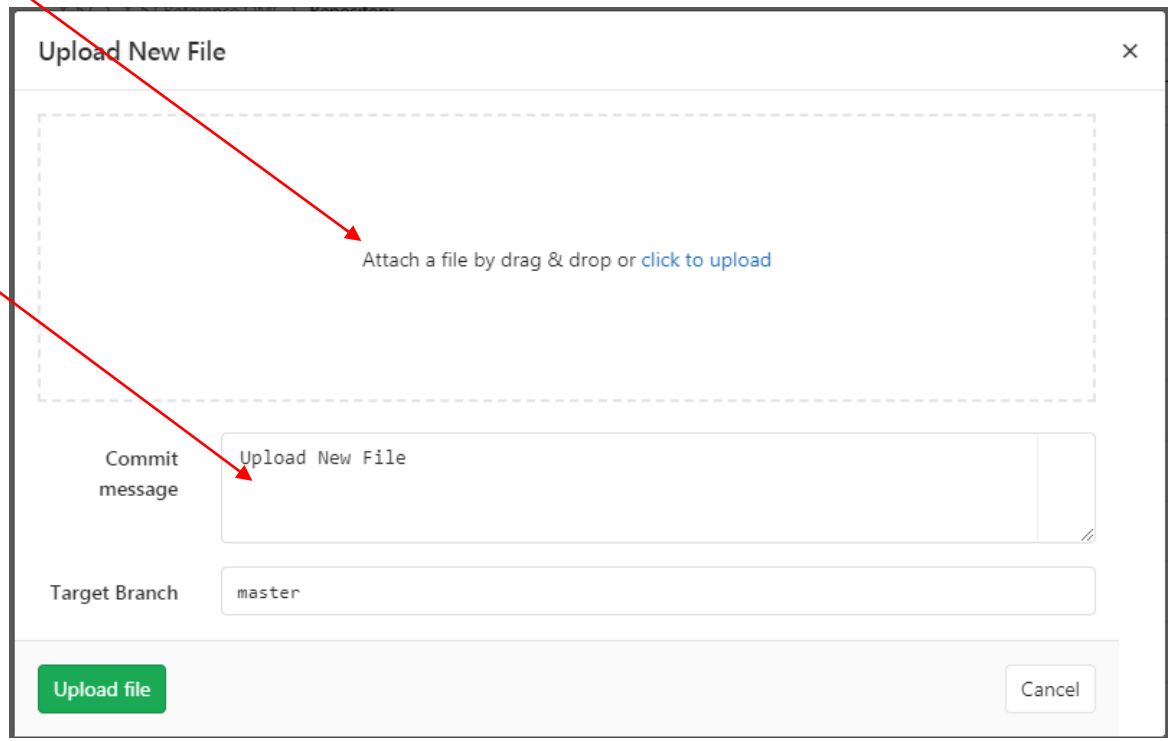
- *All without using command line.*
- *Some GitHub Enterprise systems have increased in price but may now hold SBU/ITAR code.*
  - *Your experience may differ*

GitHub / GitLab

GitLab (gitlab.com): Online collaboration and backup version control

- Can add saved files
  - New versions of existing files or add new files.
- Perform Commits

- Track Project To-Do Lists
- Manage Merge Requests
- Similar to GitHub

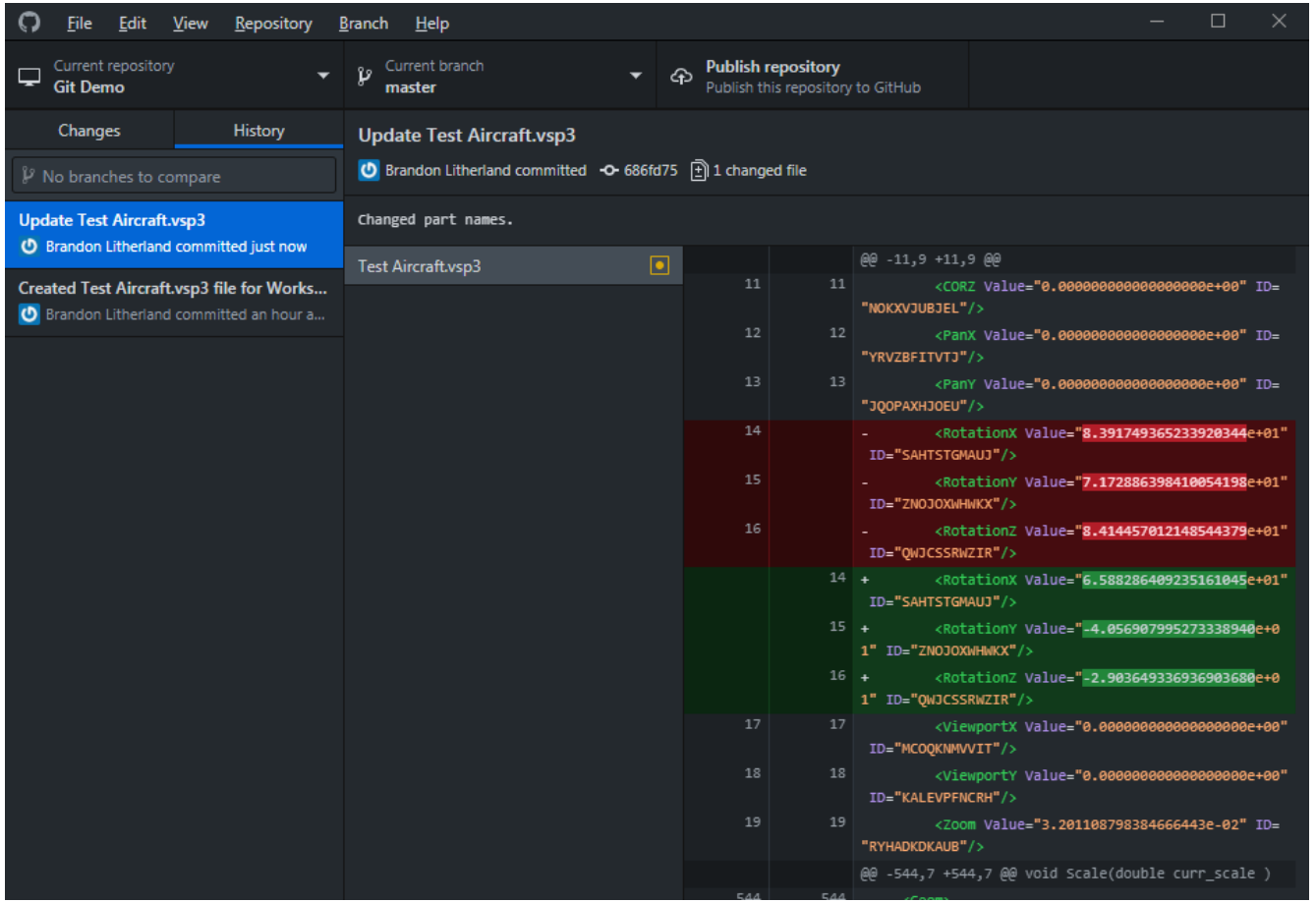


GitHub Desktop

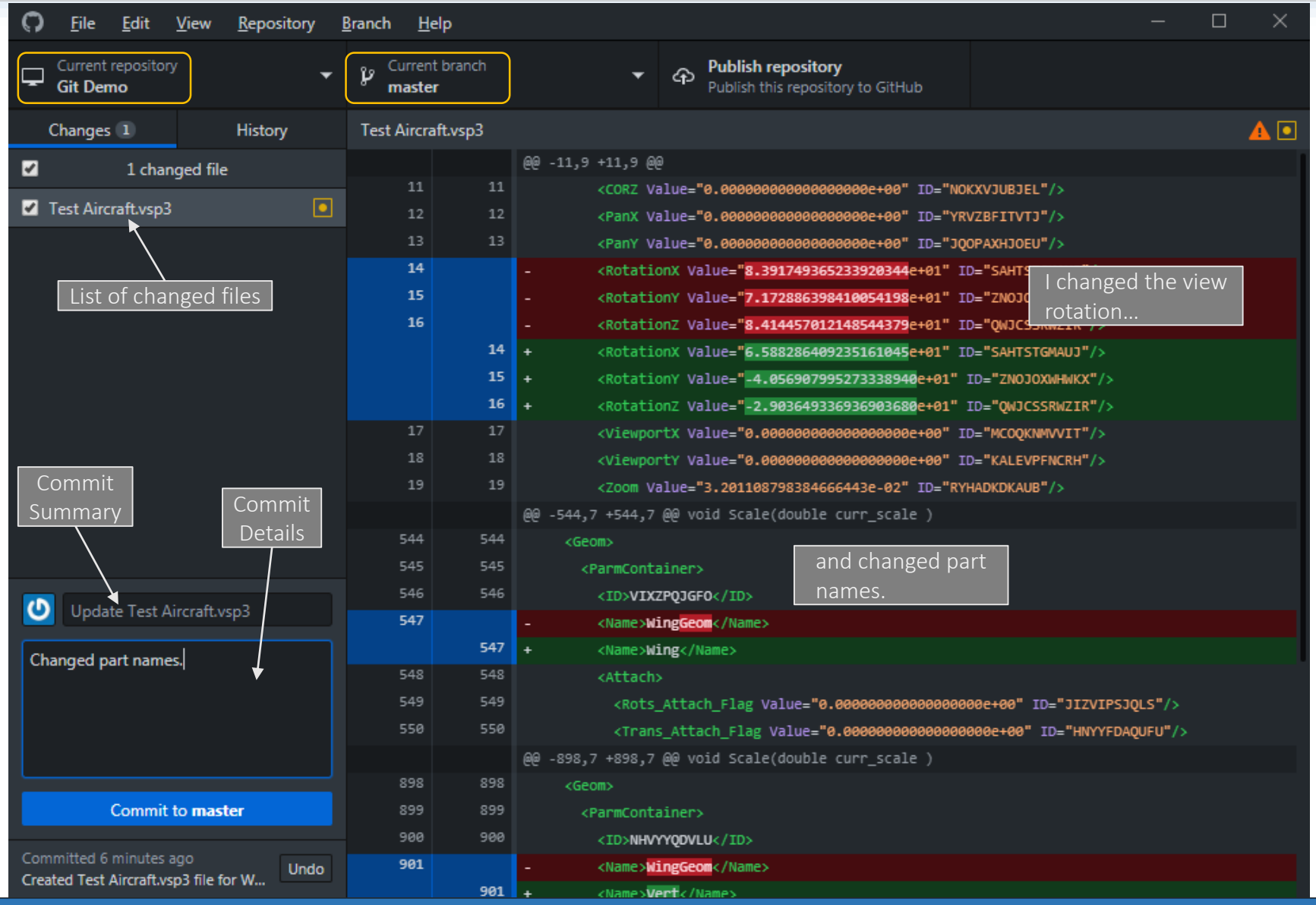
Making your life easier…

- Most of the git-bash command line operations in a handy GUI!
- Quick navigation of repository.
- Automatic visualization of version differences.
- Found here:

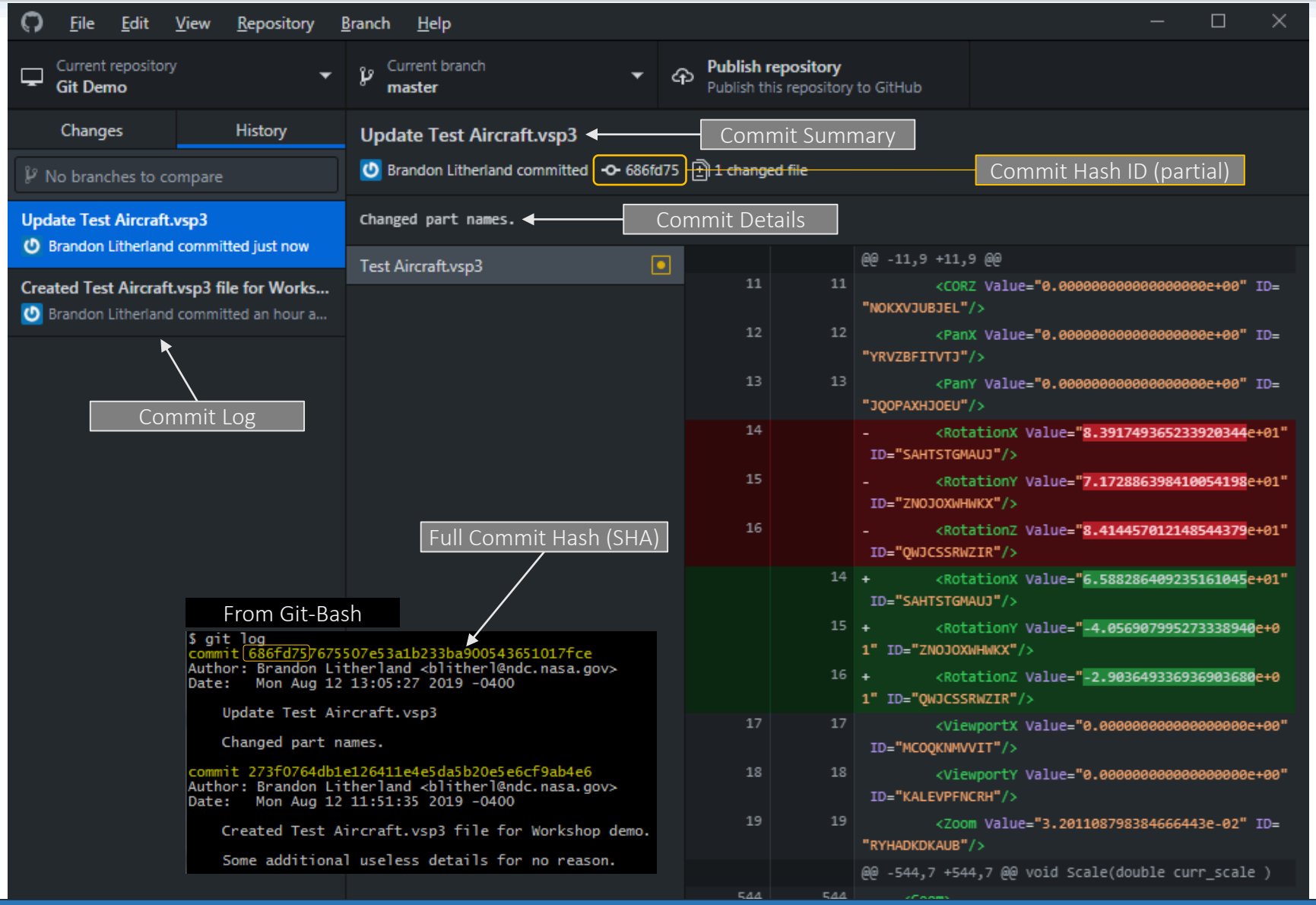
  https://desktop.github.com/



Git Desktop Example



Git Desktop Example



Demo

1. Example Repository
2. Open Model
3. Change and Save
4. View changes in Git
5. Stage > Commit
6. View History

Thank you!

Questions?

BRANDON LITHERLAND, AST
NASA LANGLEY RESEARCH CENTER
AERONAUTICS SYSTEMS ANALYSIS BRANCH