Git/GitHub Refresher

Observatory Operations Documentation

Presentation Outline

1. General Overview of Git and GitHub
    a. Why is it useful for documentation?
    b. What are git commands, and how do they work?
    c. How does one navigate between a local computer and GitHub?

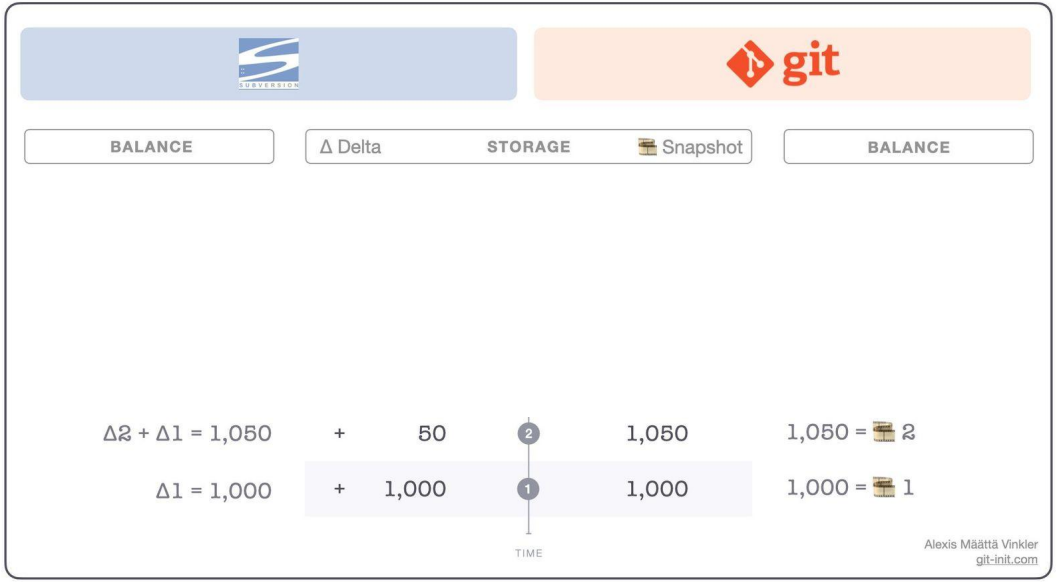
2. Documentation Workflow
    a. Creating a Jira Ticket
    b. Creating and/or Updating Documents
    c. Pull Requests & Merges

What is Git?

- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system

What is Git?

- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system



What is Git?

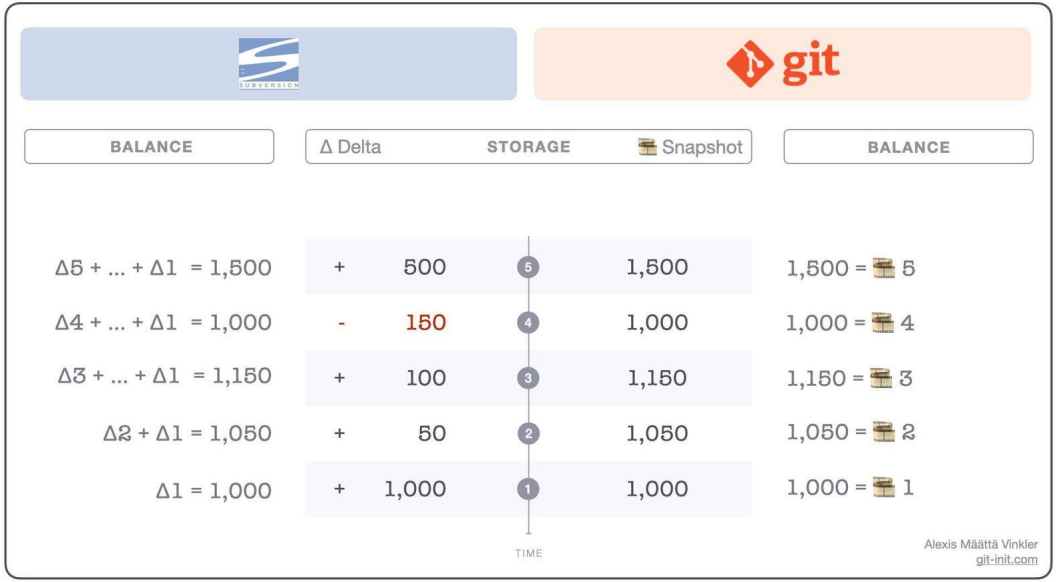
- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system

What is Git?

- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system

What is Git?

- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system



What is Git?

- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system

– *Benefits*: (1) Retrieval & (2) Comparison

What is Git?

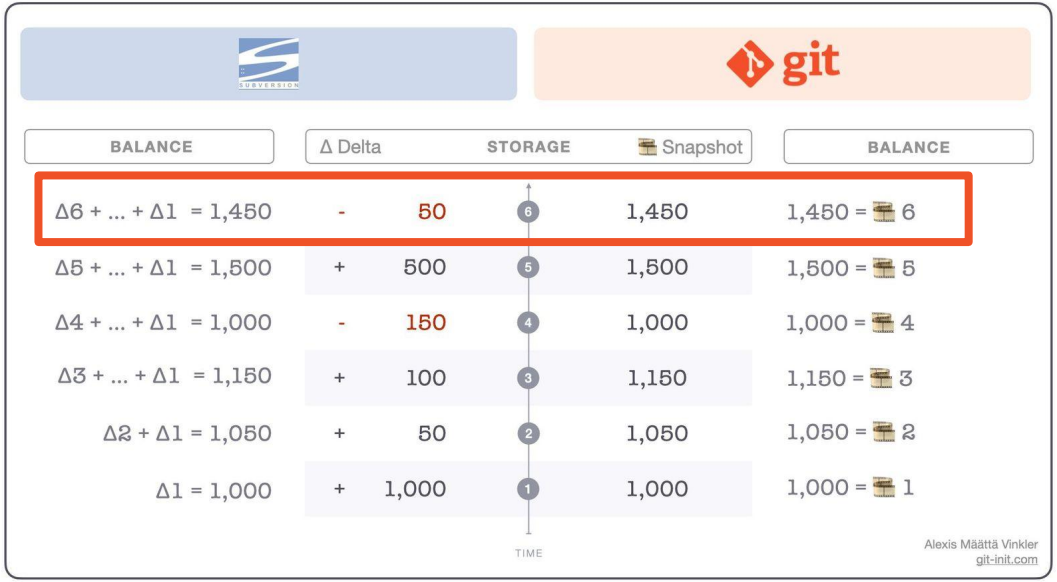
- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system

– *Benefits*: (1) Retrieval & (2) Comparison

(1) Retrieve Balance (Day 6)

Δ6 + Δ5 + Δ4 + Δ3 + Δ2 + Δ1 = **1,450**

🎞6 = **1,450**



What is Git?

- Version Control Software (VCS)
- Uses *snapshots* **not** *differences*
  - Snapshots: saves **entire state** of a file or system as a single instance
  - Delta-Based: saves **only the changes** made to a file or system

– *Benefits*: (1) Retrieval & (2) Comparison

(1) Retrieve Balance (Day 6)

Δ6 + Δ5 + Δ4 + Δ3 + Δ2 + Δ1 = **1,450**

🎞6 = **1,450**

(2) Compare Changes (Days 4 & 2)

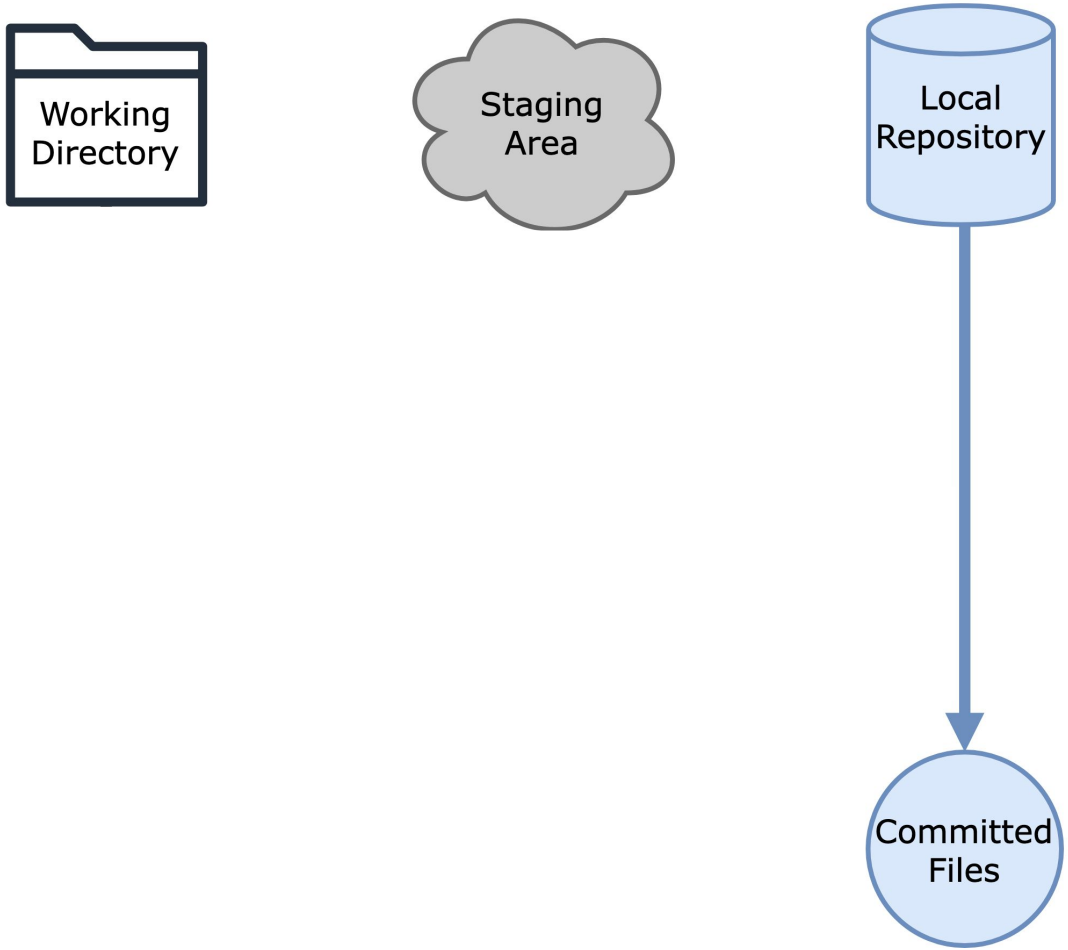
(Δ4 + Δ3 + Δ2 + Δ1) – (Δ2 + Δ1) = **– 50**

🎞4 – 🎞2 = **– 50**

How to Navigate Locally with Git

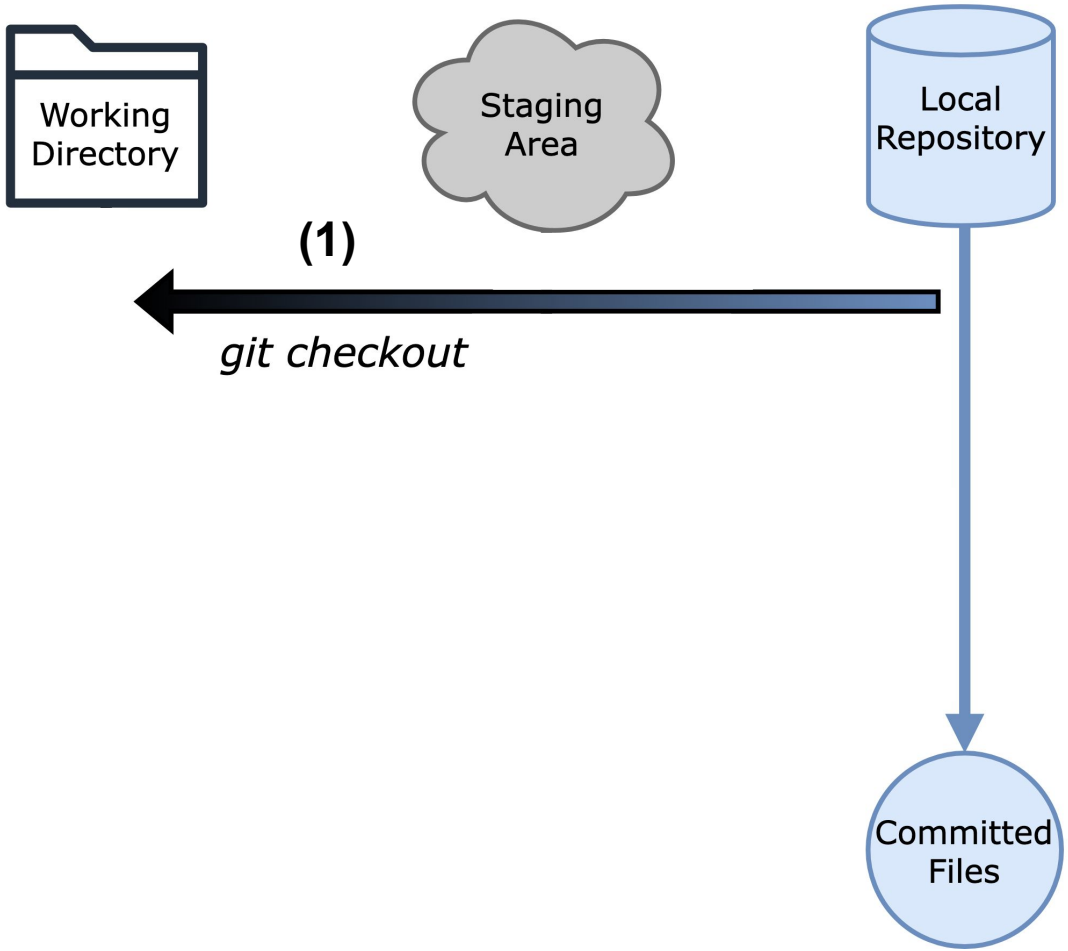
- Git primarily operates on your local computer via terminal commands
- Three main states that your files can reside in: **modified**, **staged**, and **committed**:
    - Modified: changed the file but have not committed to database.
    - Staged: marked modified file to go into next commit snapshot.
    - Committed: data safely stored in local database (*repository*).

How to Navigate Locally with Git



Common Workflow

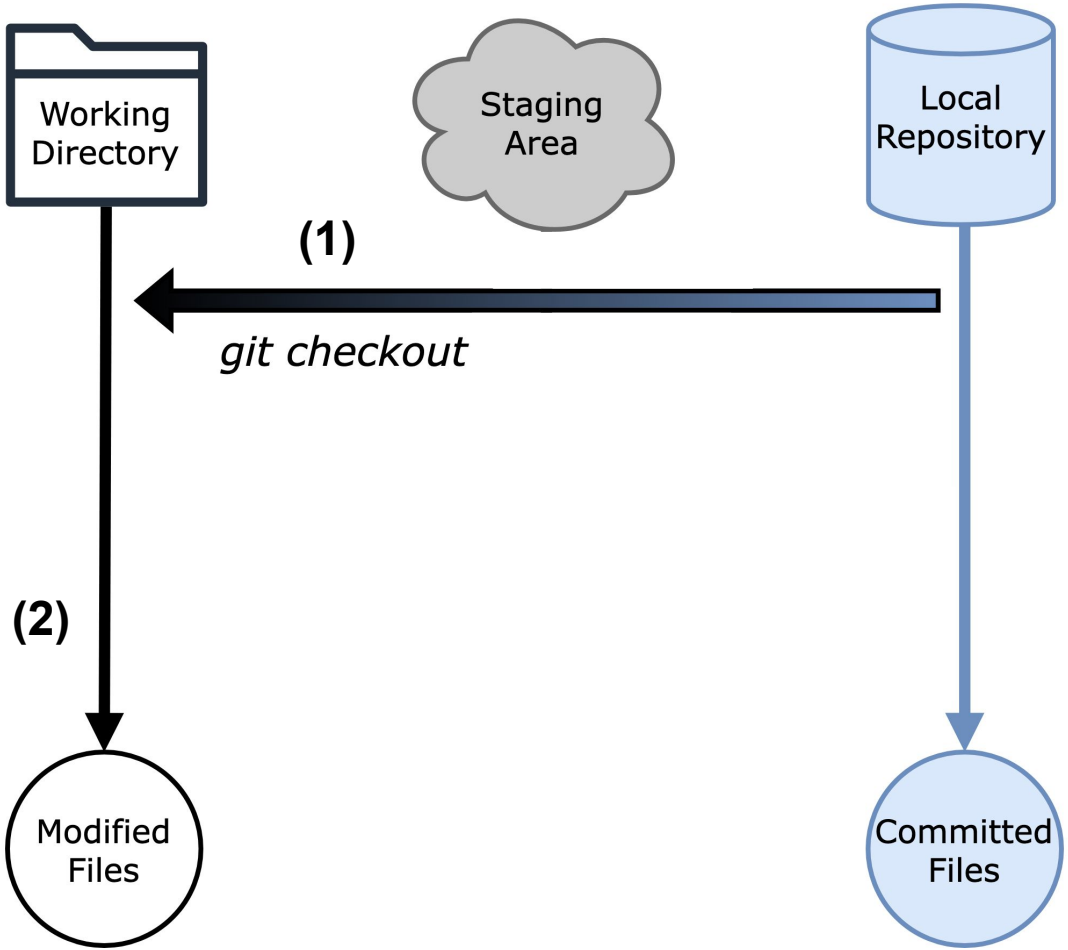
How to Navigate Locally with Git



Common Workflow

(1) Access the files you wish to change: **`git checkout`**

How to Navigate Locally with Git

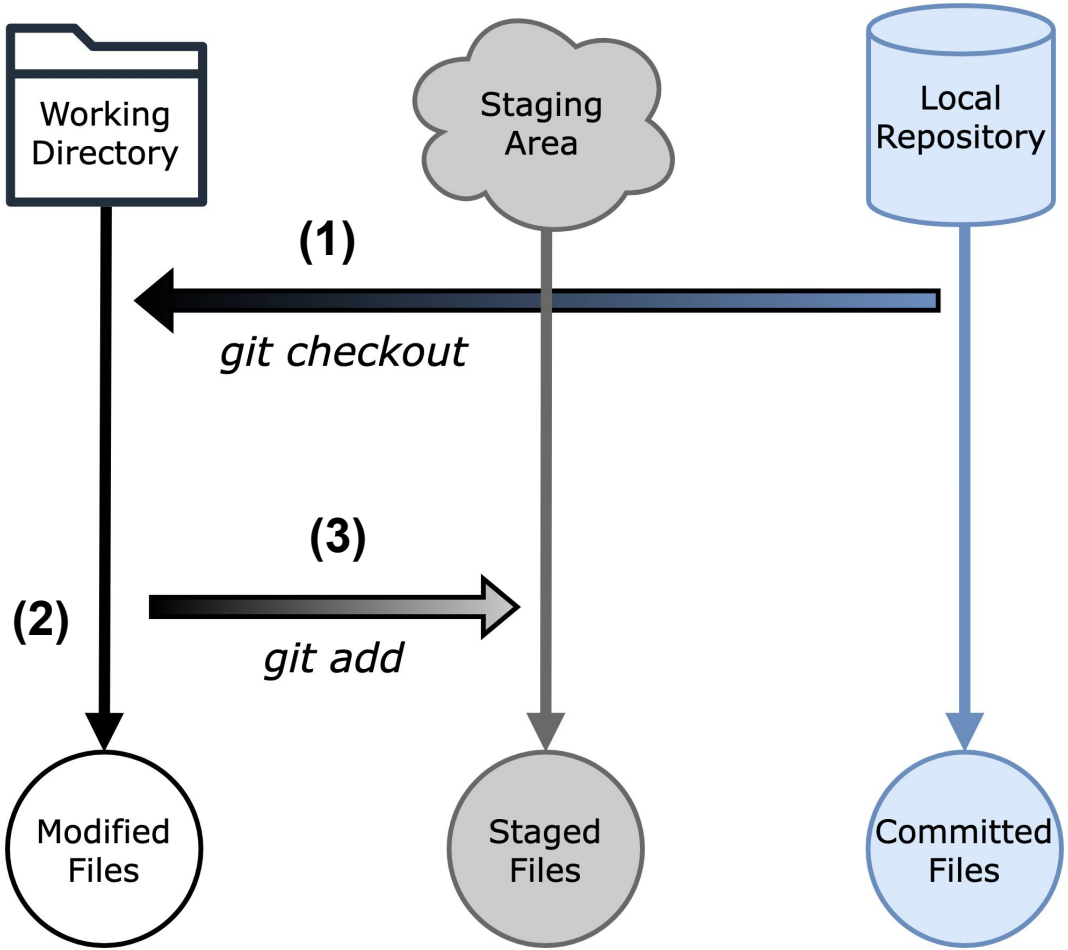


Common Workflow

(1) Access the files you wish to change: **`git checkout`**

(2) Modify files in your directory.

How to Navigate Locally with Git



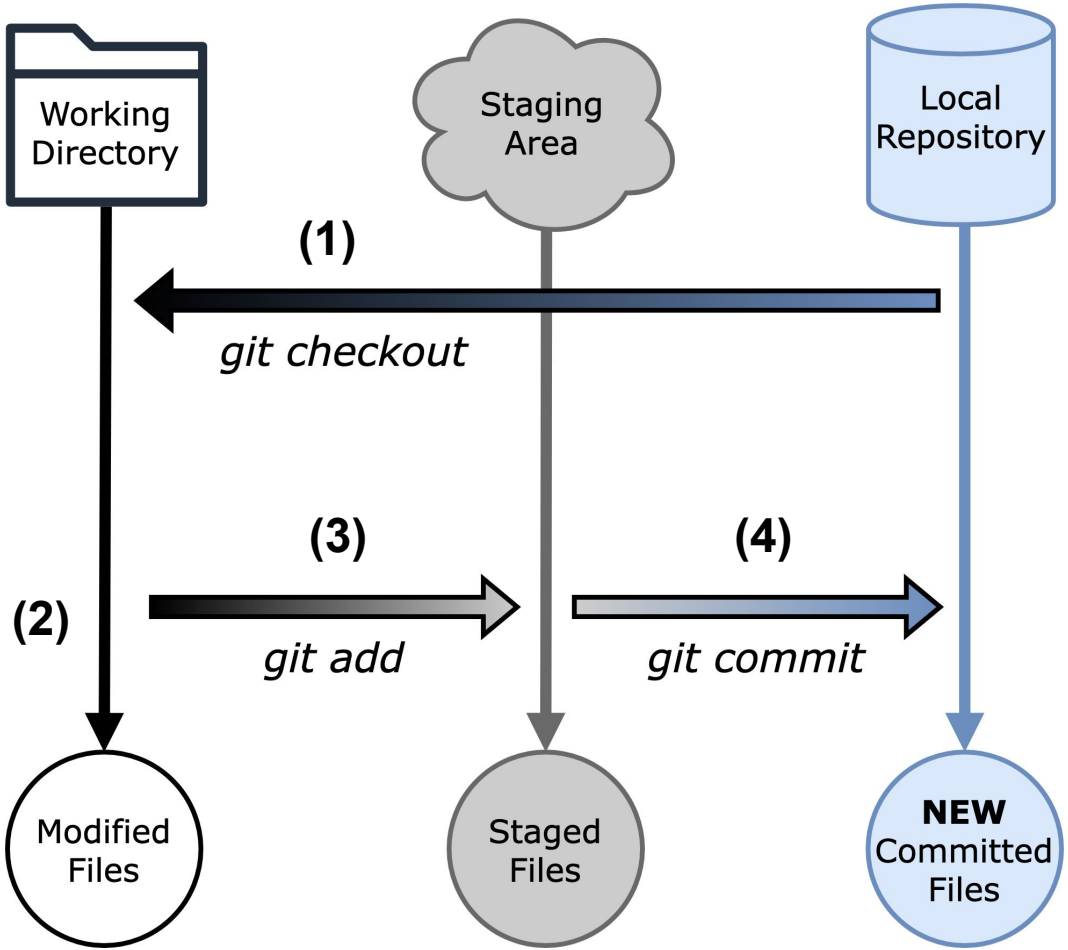
Common Workflow

(1) Access the files you wish to change: **`git checkout`**

(2) Modify files in your directory.

(3) Select changes you want to be permanent: **`git add <files>`**

How to Navigate Locally with Git



Common Workflow

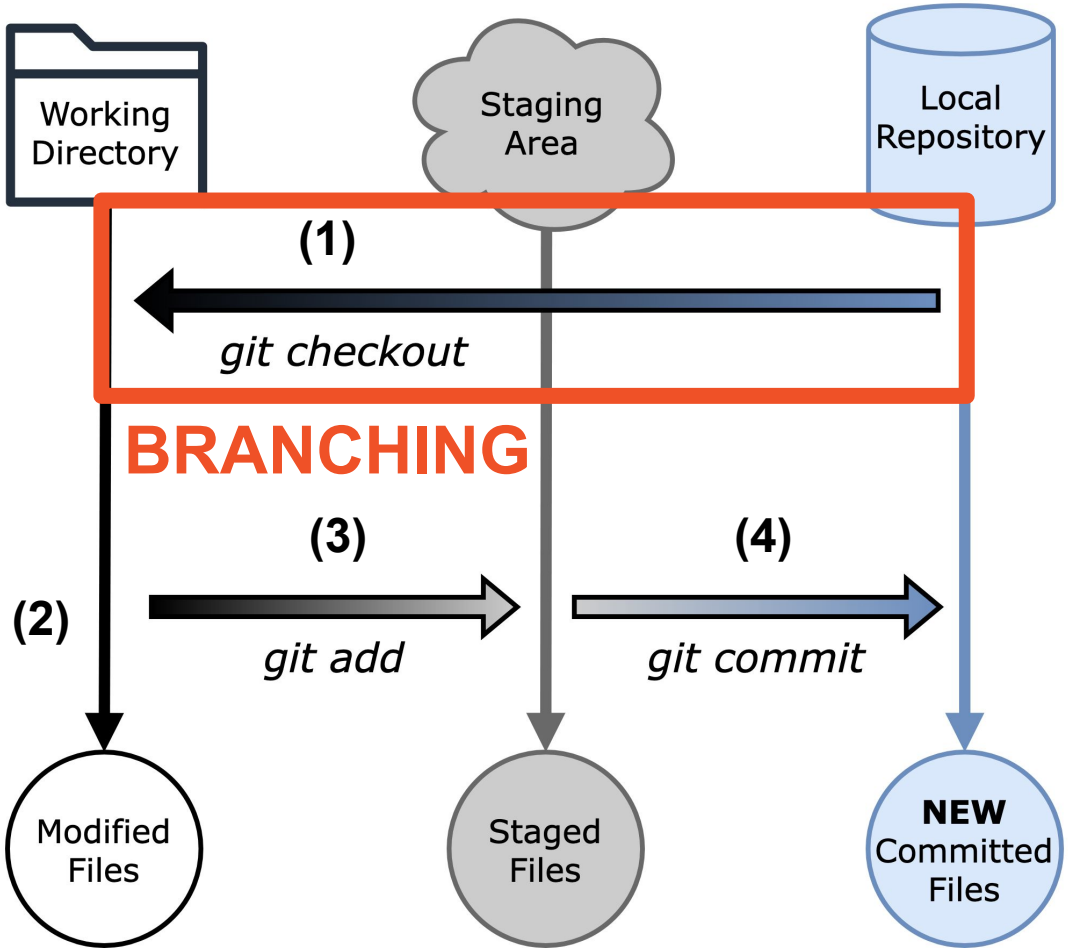
(1) Access the files you wish to change: **`git checkout`**

(2) Modify files in your directory.

(3) Select changes you want to be permanent: **`git add <files>`**

(4) Store changes permanently in Git repository: **`git commit`**

How to Navigate Locally with Git



Common Workflow

(1) Access the files you wish to change: **`git checkout`**

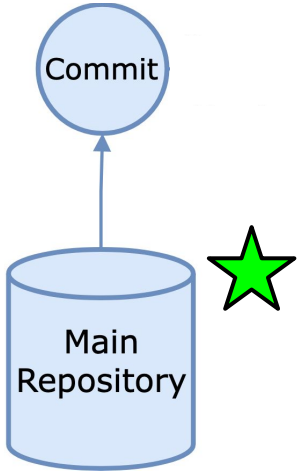
(2) Modify files in your directory.

(3) Select changes you want to be permanent: **`git add <files>`**

(4) Store changes permanently in Git repository: **`git commit`**

Accessing Branches with Git

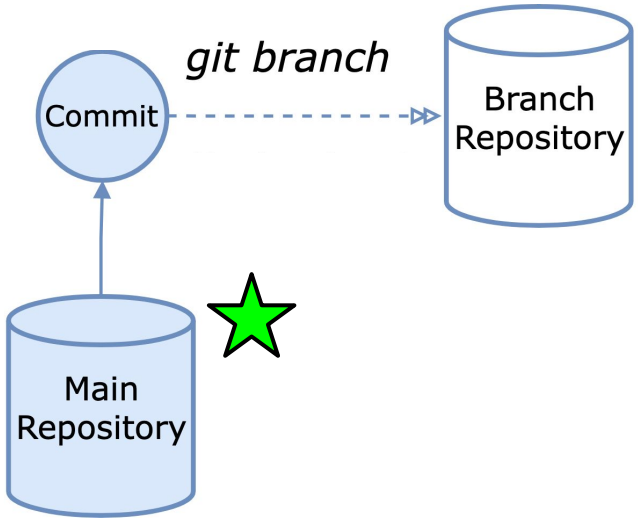
- Branching allows **safe** version control.
    - Creates a copy of repository.



Accessing Branches with Git

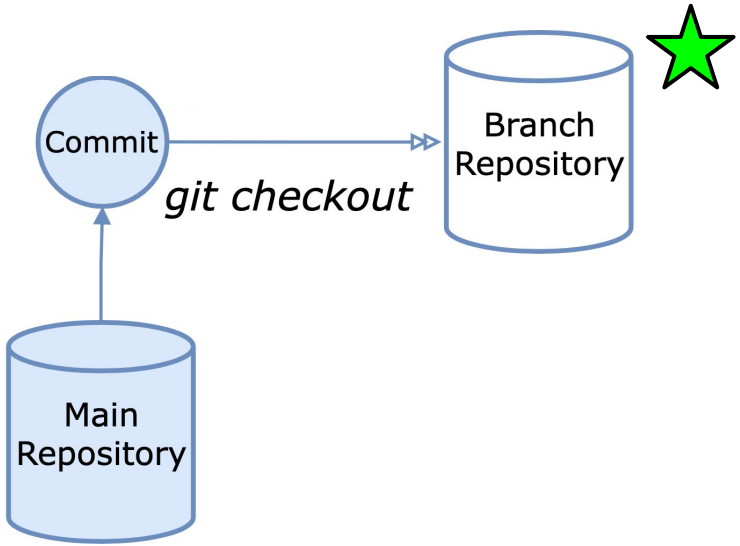
- Branching allows **safe** version control.
  - Creates a copy of repository.

- To make a new branch:
  - `git branch <branch>`



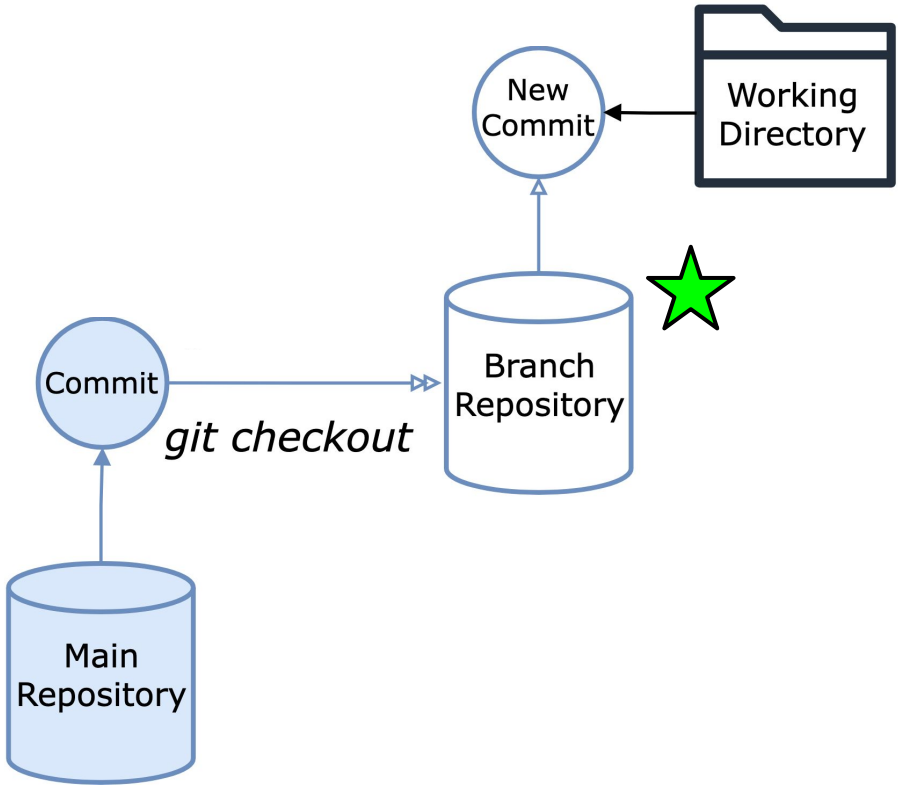
Accessing Branches with Git

- Branching allows **safe** version control.
  - Creates a copy of repository.
- To make a new branch:
  - `git branch <branch>`
- To access different branches:
  - `git checkout <branch>`



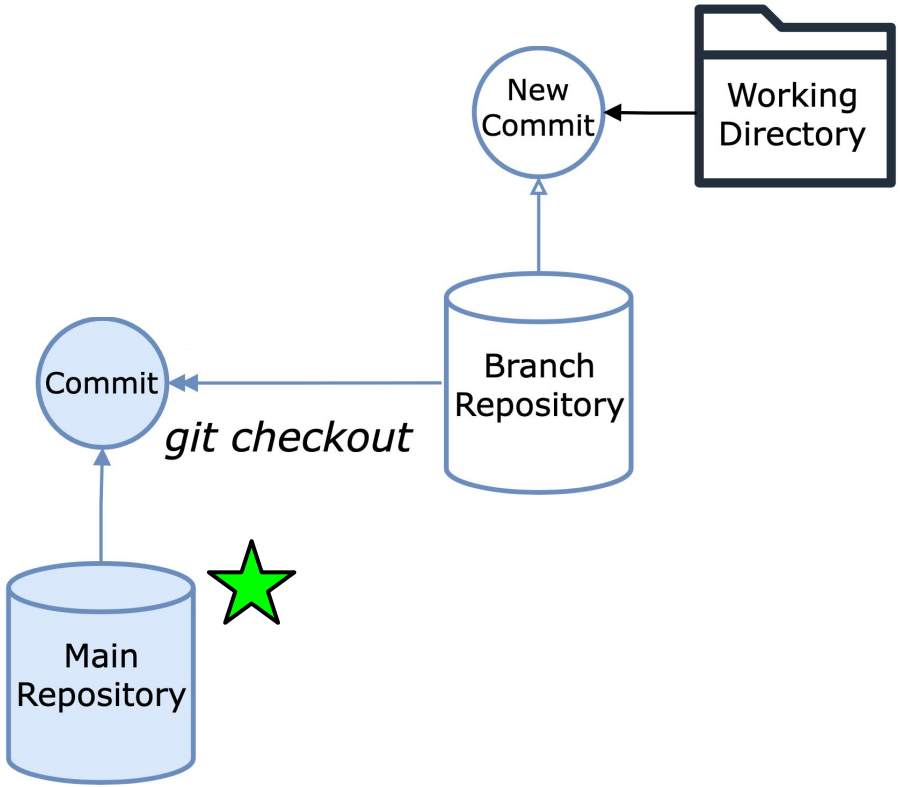
Accessing Branches with Git

- Branching allows **safe** version control.
  - Creates a copy of repository.
- To make a new branch:
  - `git branch <branch>`
- To access different branches:
  - `git checkout <branch>`
- Once change are made, add them back into the “main” repository:



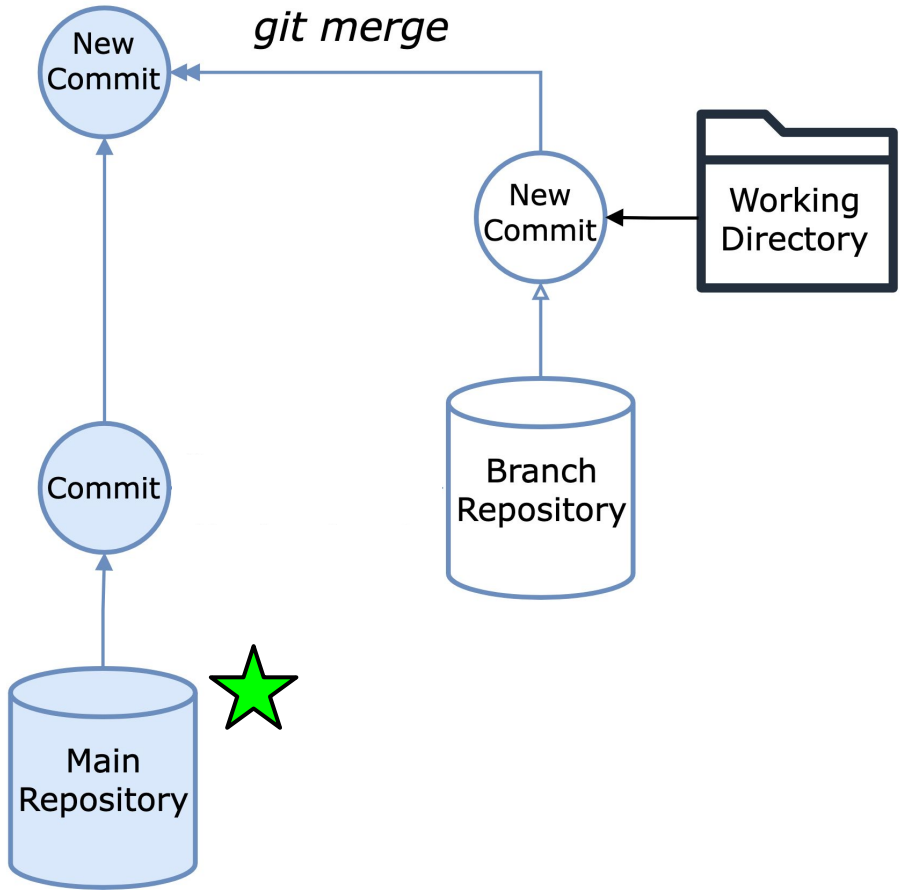
Accessing Branches with Git

- Branching allows **safe** version control.
  - Creates a copy of repository.
- To make a new branch:
  - `git branch <branch>`
- To access different branches:
  - `git checkout <branch>`
- Once change are made, add them back into the “main” repository:
  - `git checkout <main>`



Accessing Branches with Git

- Branching allows **safe** version control.
  - Creates a copy of repository.
- To make a new branch:
  - `git branch <branch>`
- To access different branches:
  - `git checkout <branch>`
- Once change are made, add them back into the “main” repository:
  - `git checkout <main>`
  - `git merge <branch>`



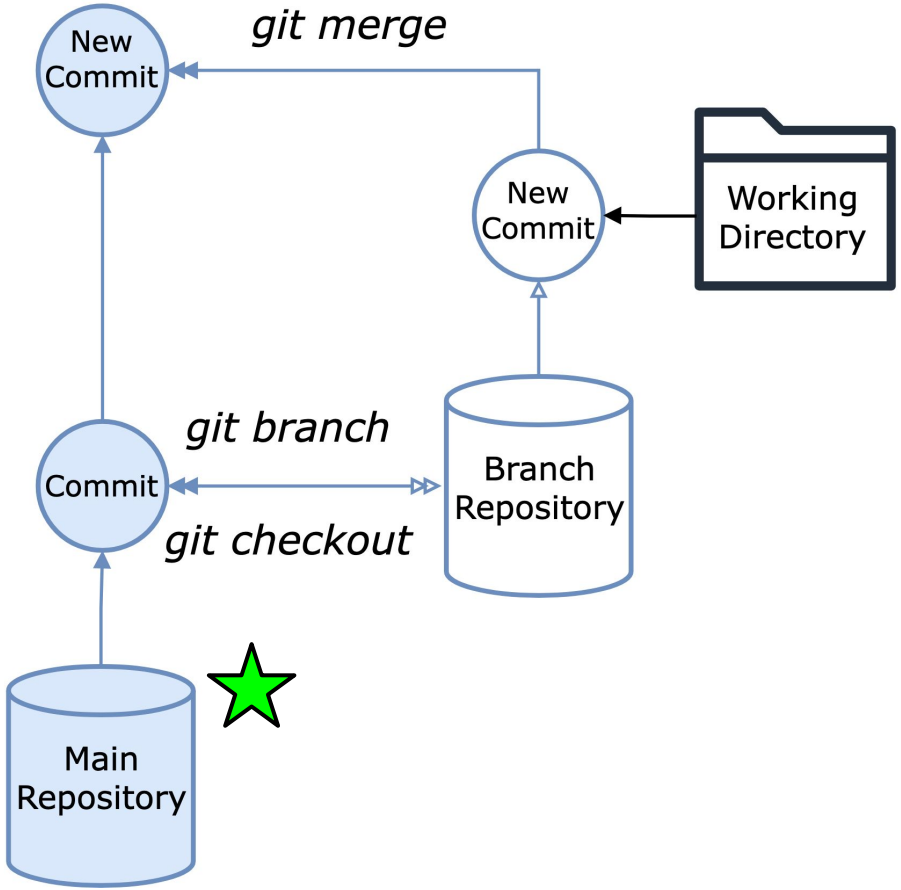
Accessing Branches with Git

IMPORTANT!

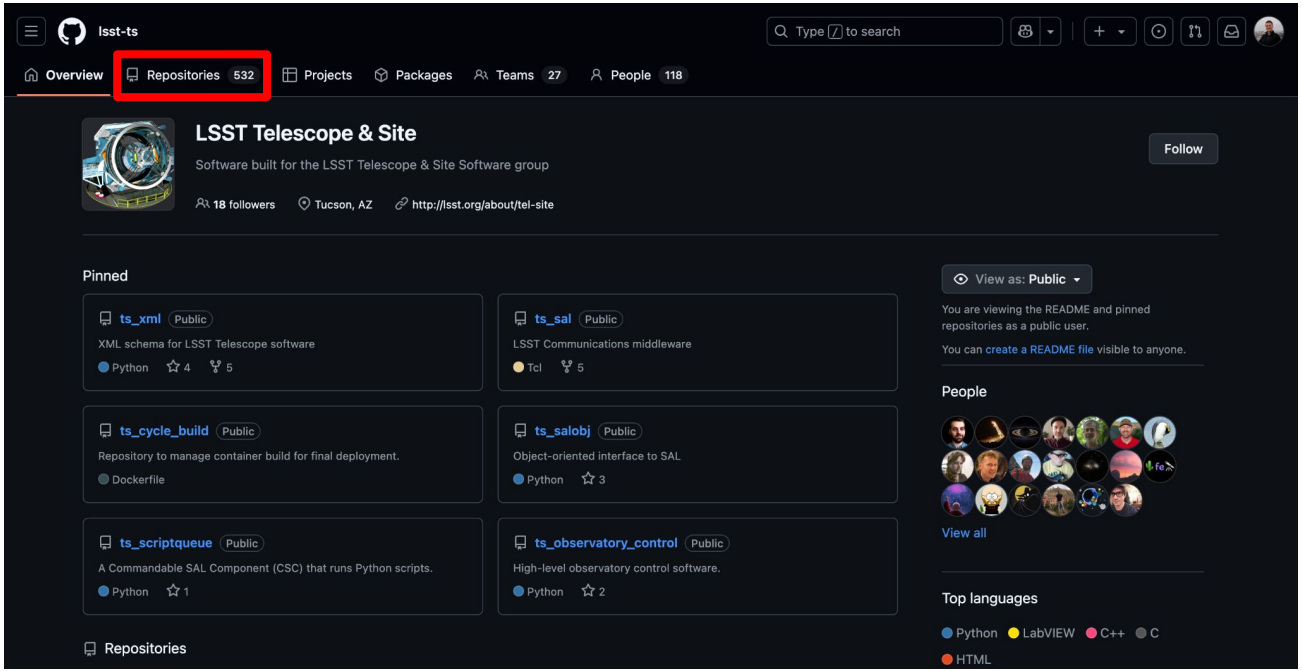
Pay attention to the branch you are on!

– Make edits on **Branch Repository**

– Branch/Merge on **Main Repository**



What is GitHub?



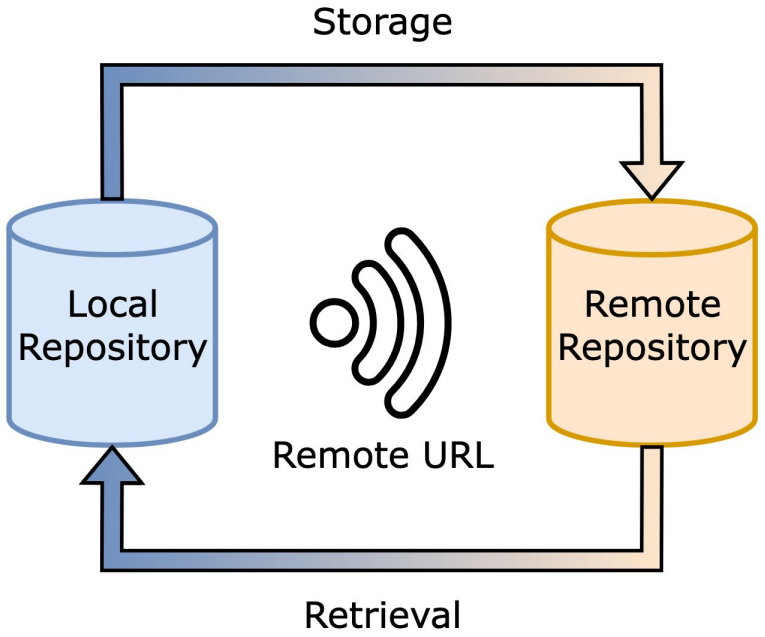
- Cloud-based *Git repository* hosting service.
- Easier for <u>individuals and teams</u> to use Git for **version control** and **collaboration**.
    - More user-friendly than Git alone.
    - Free, public resource for open-source projects.
    - Well-documented reviewing procedures.

Connecting Local with GitHub

- GitHub stores files in an online database called a **remote repository**.

- A remote URL allows a connection point between local and remote repositories.
  a. HTTPS: `https://github.com/user/repo.git`
  b. SSH: `git@github.com:user/repo.git`

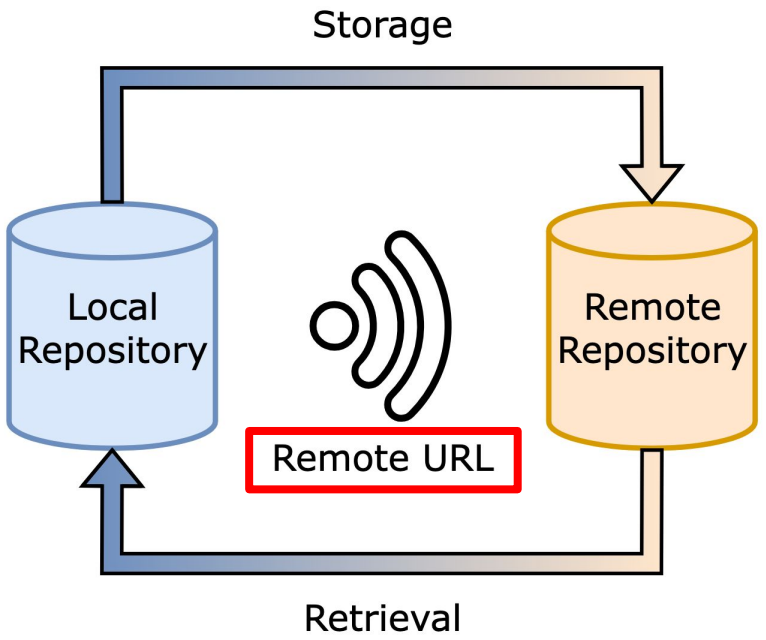
- Connecting local computer with GitHub allows easy storage and retrieval of files across different repositories.



Connecting Local with GitHub

Remote URL

1. Create: `git init`
   `git remote add origin <URL>`
2. Copy: `git clone <URL>`
3. Check: `git remote -v`
4. Change: `git remote set-url origin <URL>`



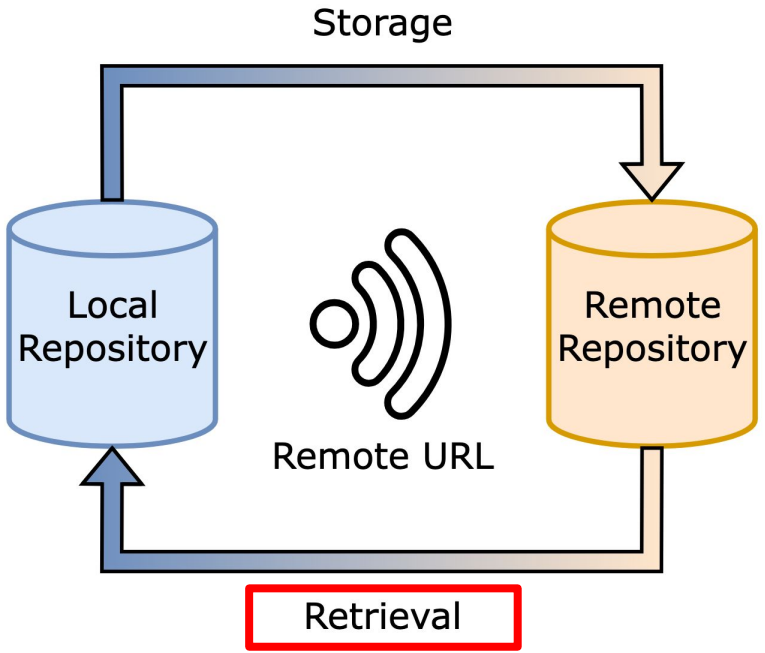
Connecting Local with GitHub

Remote URL

1. Create: `git init`
   `git remote add origin <URL>`
2. Copy: `git clone <URL>`
3. Check: `git remote -v`
4. Change: `git remote set-url origin <URL>`

Retrieval

1. Local Repository: `git fetch`
2. Working Directory: `git pull`



Connecting Local with GitHub

Remote URL

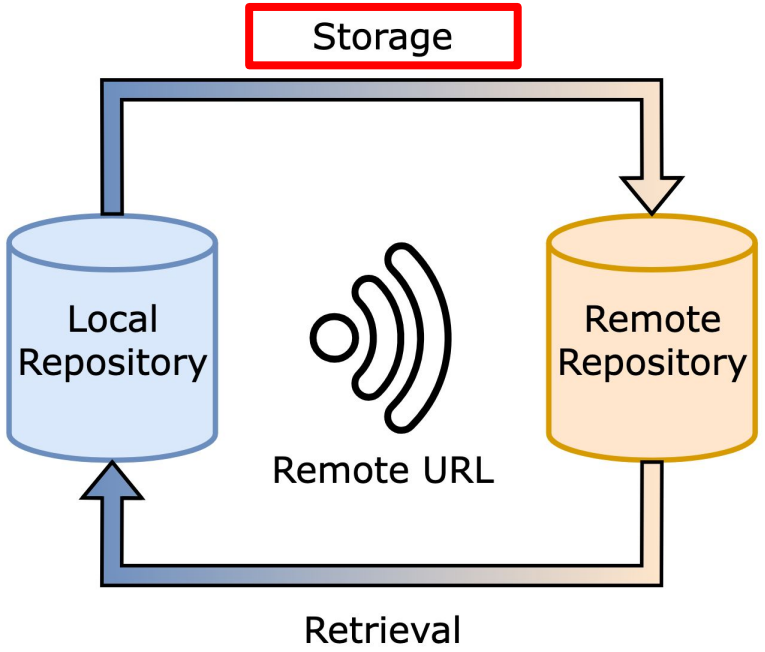
1. Create: `git init`
   `git remote add origin <URL>`
2. Copy: `git clone <URL>`
3. Check: `git remote -v`
4. Change: `git remote set-url origin <URL>`

Retrieval

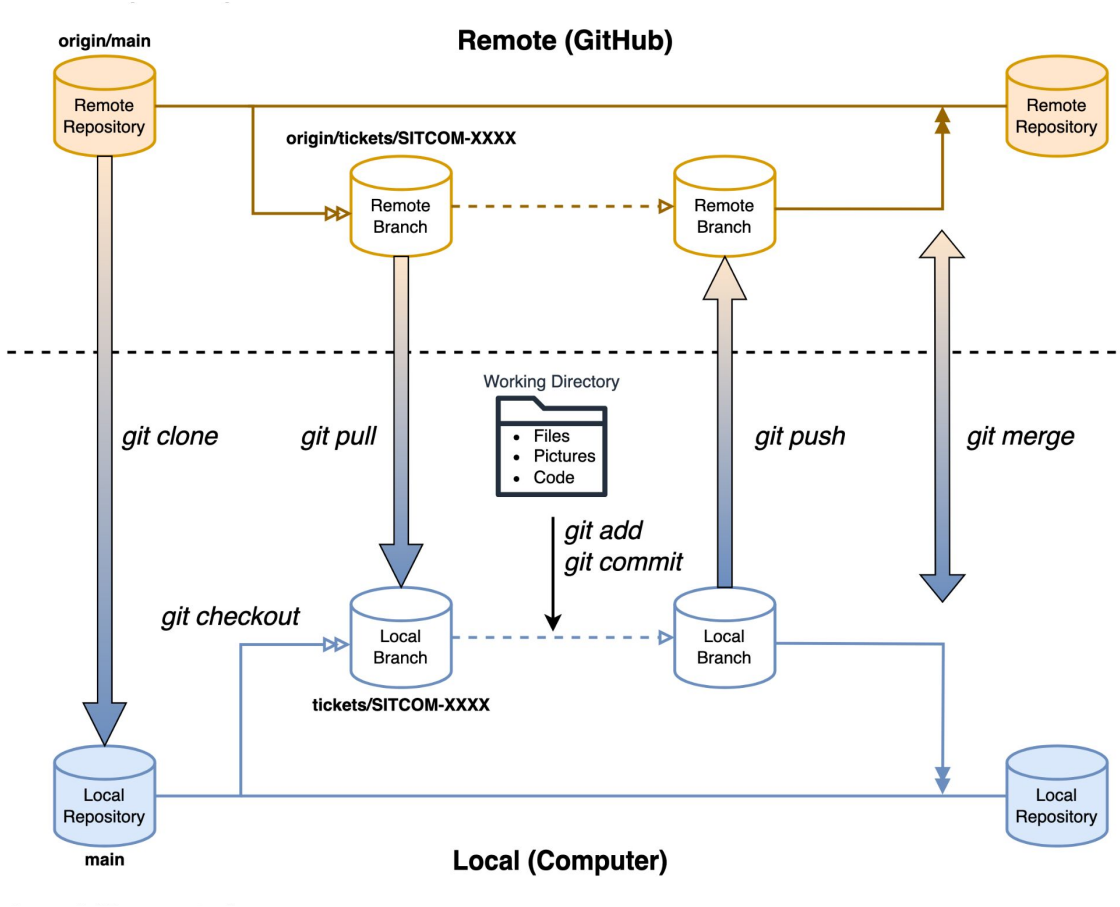
1. Local Repository: `git fetch`
2. Working Directory: `git pull`

Storage

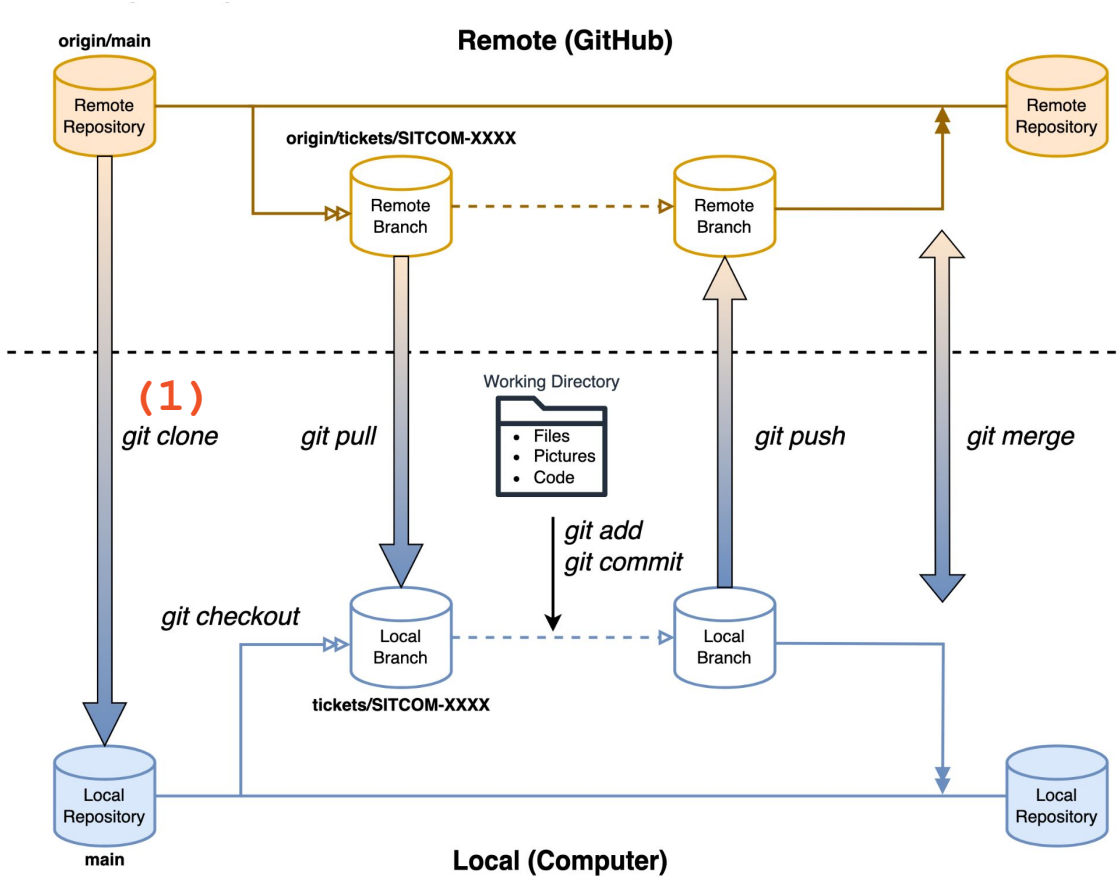
1. Remote Repository: `git push`



Overall Workflow – Documentation

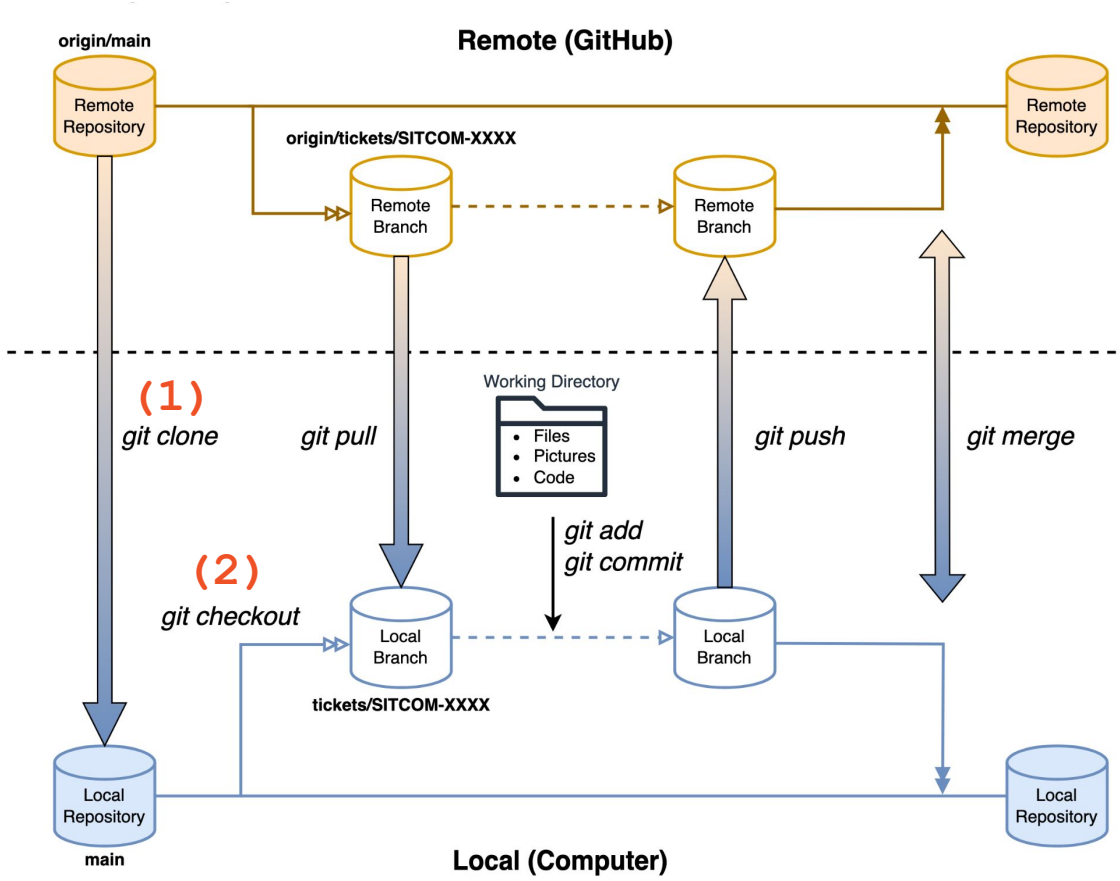


Overall Workflow – Documentation



(1) Connect to Obs-Ops repository.

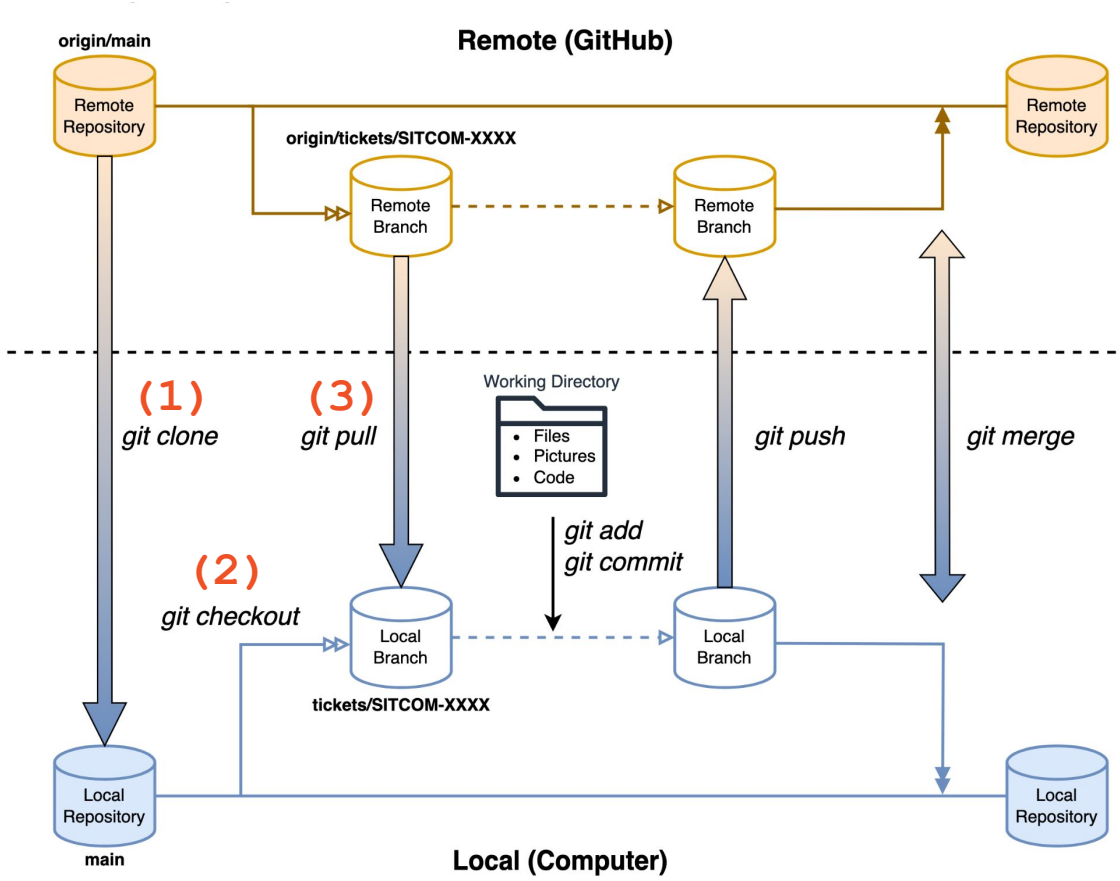
Overall Workflow – Documentation



(1) Connect to Obs-Ops repository.

(2) Create a Jira ticket branch.

Overall Workflow – Documentation

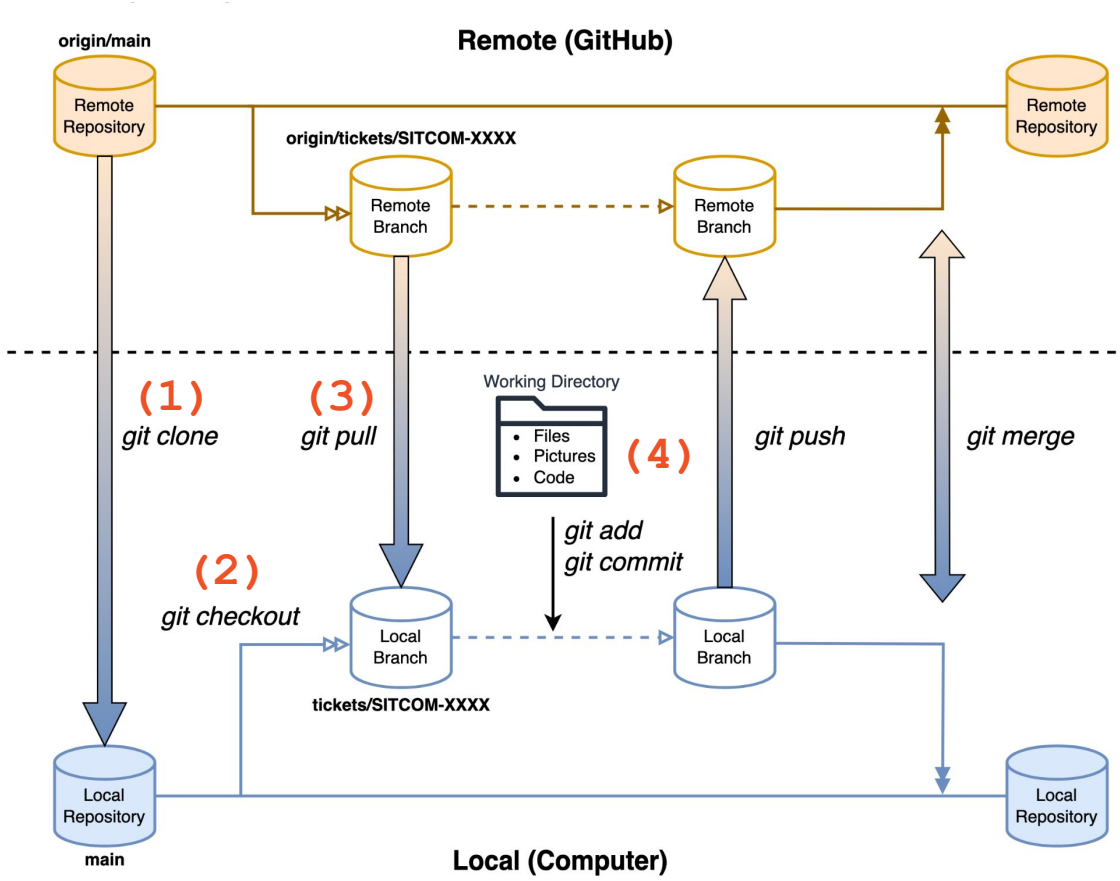


(1) Connect to Obs-Ops repository.

(2) Create a Jira ticket branch.

(3) Pull GitHub updates into branch.

Overall Workflow – Documentation



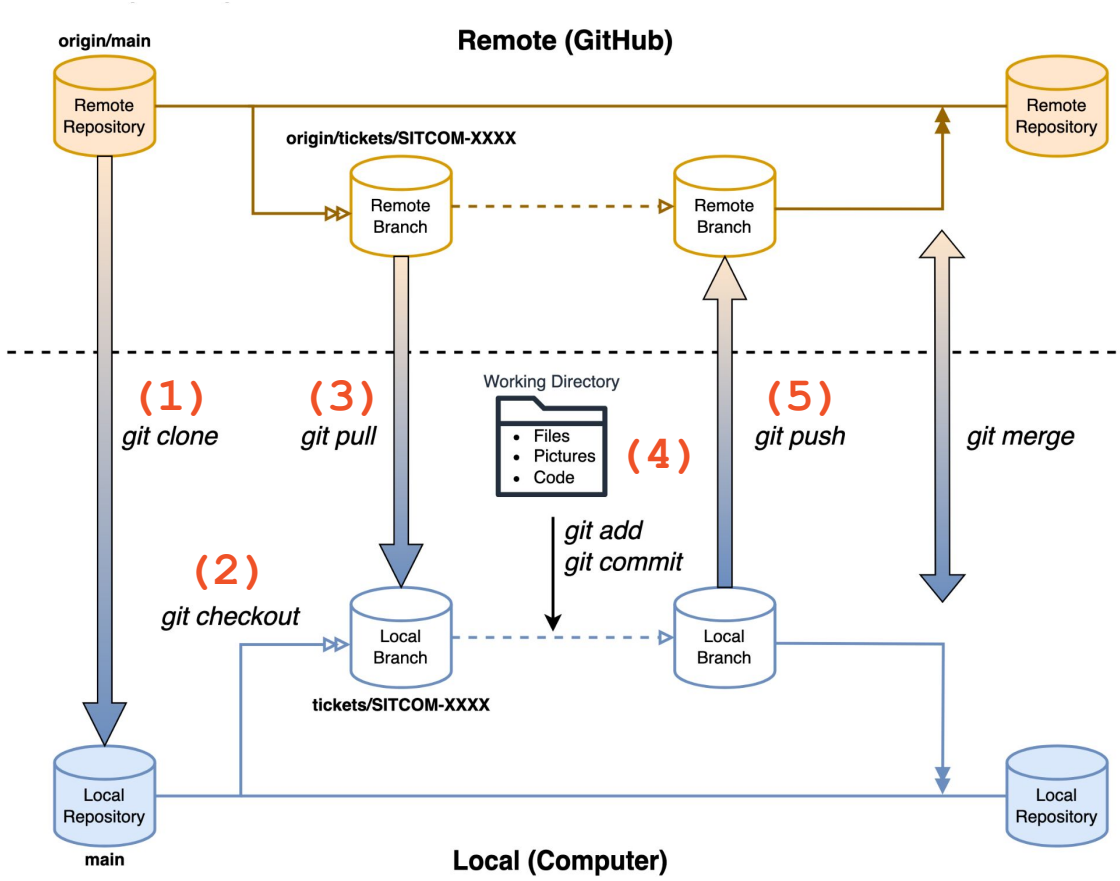
(1) Connect to Obs-Ops repository.

(2) Create a Jira ticket branch.

(3) Pull GitHub updates into branch.

(4) Edit files and commit.

Overall Workflow – Documentation



(1) Connect to Obs-Ops repository.

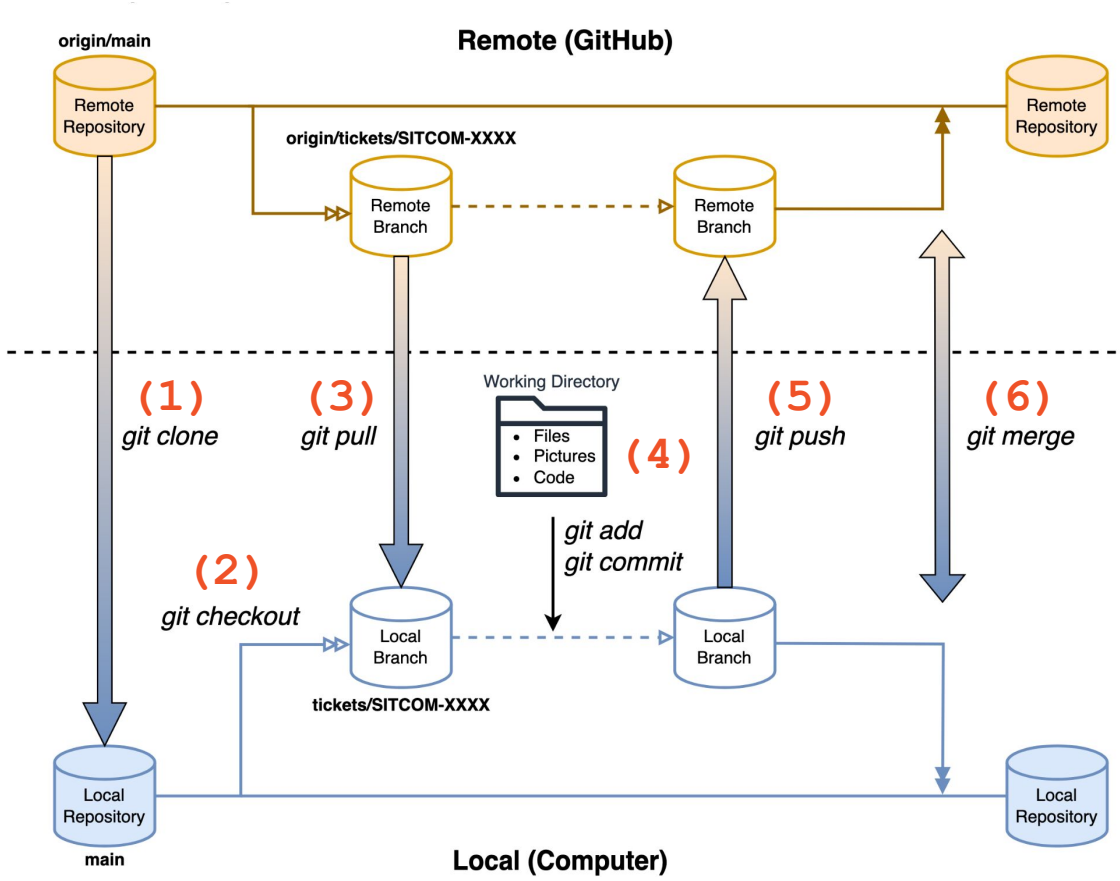
(2) Create a Jira ticket branch.

(3) Pull GitHub updates into branch.

(4) Edit files and commit.

(5) Push commits to GitHub.

Overall Workflow – Documentation



(1) Connect to Obs-Ops repository.

(2) Create a Jira ticket branch.

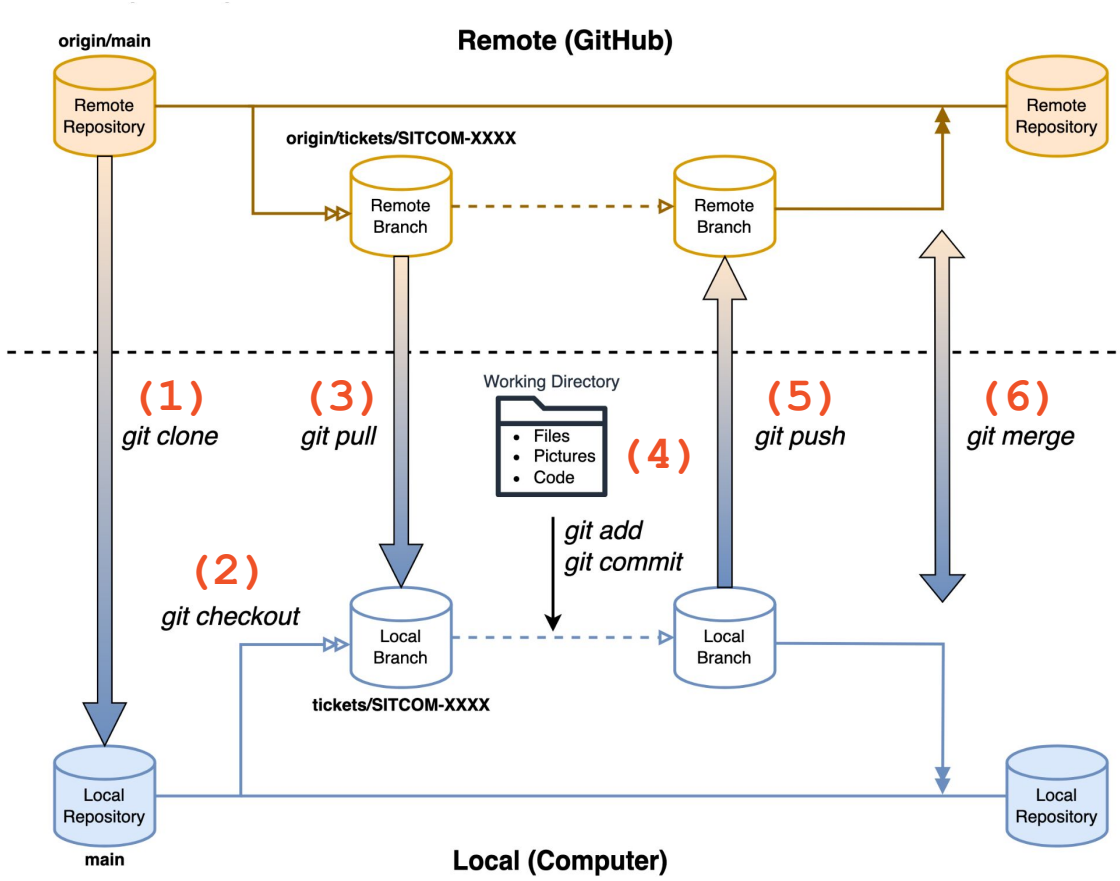
(3) Pull GitHub updates into branch.

(4) Edit files and commit.

(5) Push commits to GitHub.

(6) When ready, merge branches.

Overall Workflow – Documentation



(1) Connect to Obs-Ops repository.

(2) Create a Jira ticket branch.*

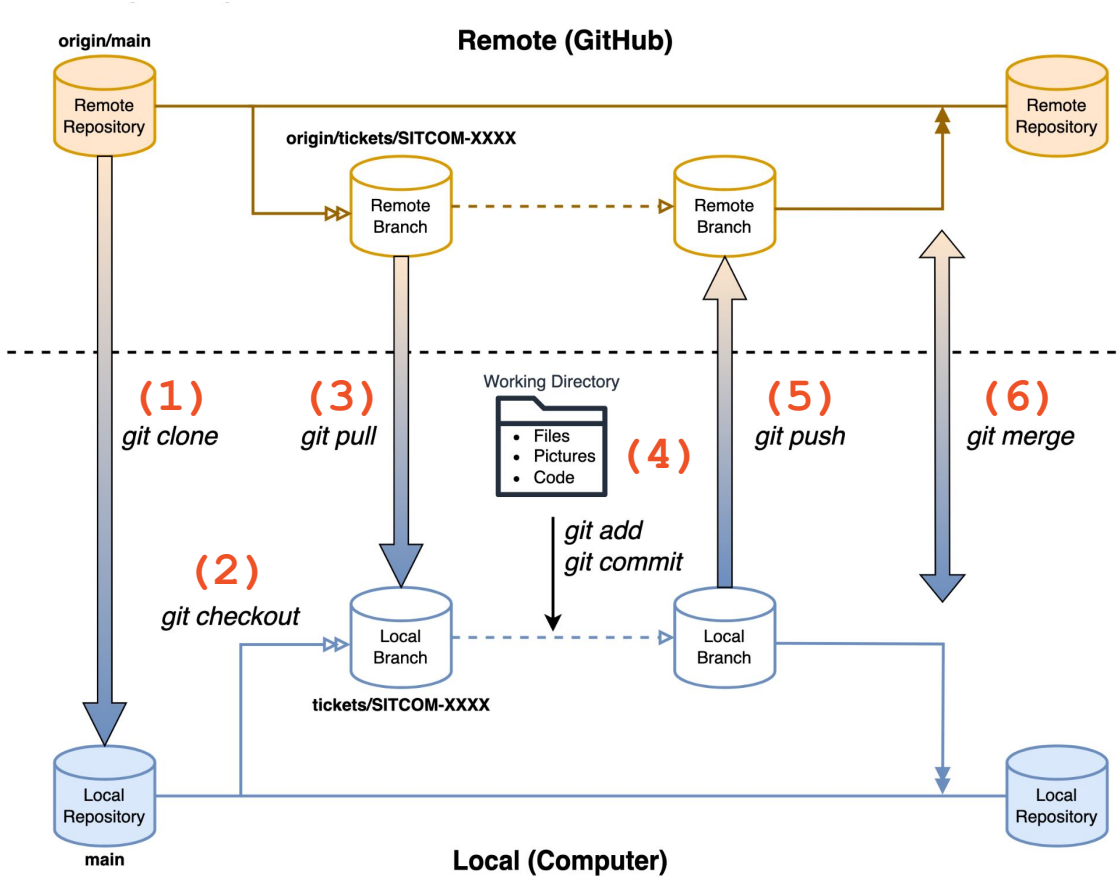
(3) Pull GitHub updates into branch.

(4) Edit files and commit.*

(5) Push commits to GitHub.

(6) When ready, merge branches.*

Overall Workflow – Documentation



(1) Connect to Obs-Ops repository.

(2) Create a Jira ticket branch.*

(3) Pull GitHub updates into branch.

(4) Edit files and commit.*

(5) Push commits to GitHub.

(6) When ready, merge branches.*

NOTE: For a more involved step-by-step approach, visit:

https://obs-ops.lsst.io/project/contributing.html

Creating a Jira Ticket Branch

Jira Ticket Layout

Project: SITCOM Work Management
Issue Type: Story
Component: SIT-Com Organizational Support
Labels: documentation
Assignee: The person that's going to write it (you or someone else).
Reviewer: Subsystem specialist/manager or one of the other members from the OS team.
Start and End Date: Estimate time interval.

- Add links to pages if applicable.

- The new Jira Ticket will have a unique 4-number identifier (e.g., ***SITCOM–1928***).
- Once you start working, make sure to track the ticket properly:

To-Do ⇒ In Progress ⇒ In Review ⇒ Reviewed ⇒ Done

Creating a Jira Ticket Branch

Jira Ticket Layout

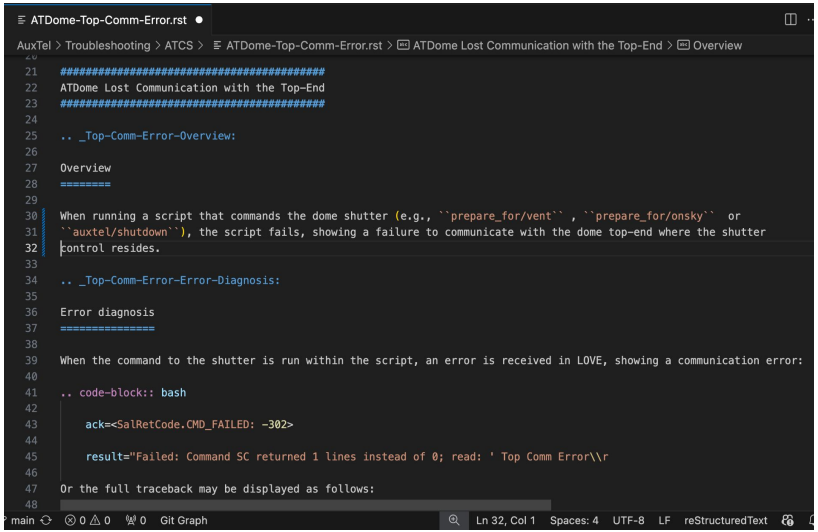
Project: SITCOM Work Management
Issue Type: Story
Component: SIT-Com Organizational Support
Labels: documentation
Assignee: The person that's going to write it (you or someone else).
Reviewer: Subsystem specialist/manager or one of the other members from the OS team.
Start and End Date: Estimate time interval.

- Add links to pages if applicable.

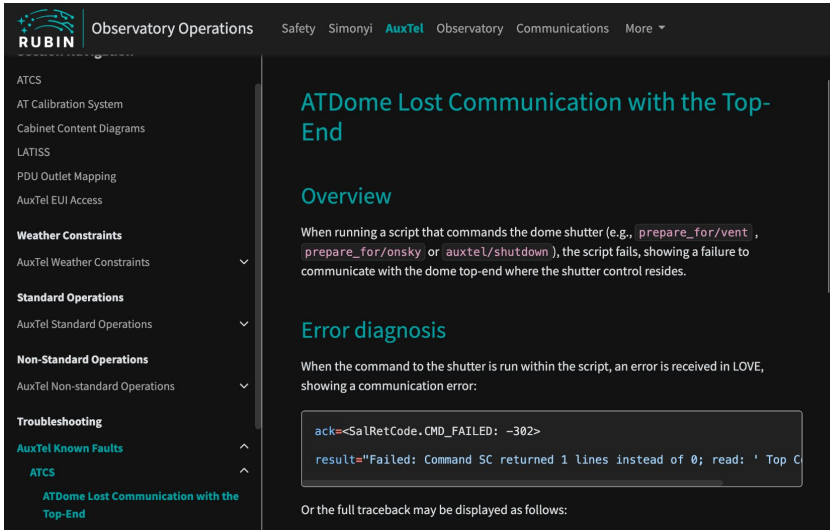
1. Create ticket branch (local): `git branch tickets/SITCOM-1928`
2. Move to branch: `git checkout tickets/SITCOM-1928`
3. After first new commits, establish remote branch:
   `git push --set-upstream origin tickets/SITCOM-1928`

Editing Files – reStructuredText (reST)

reST File



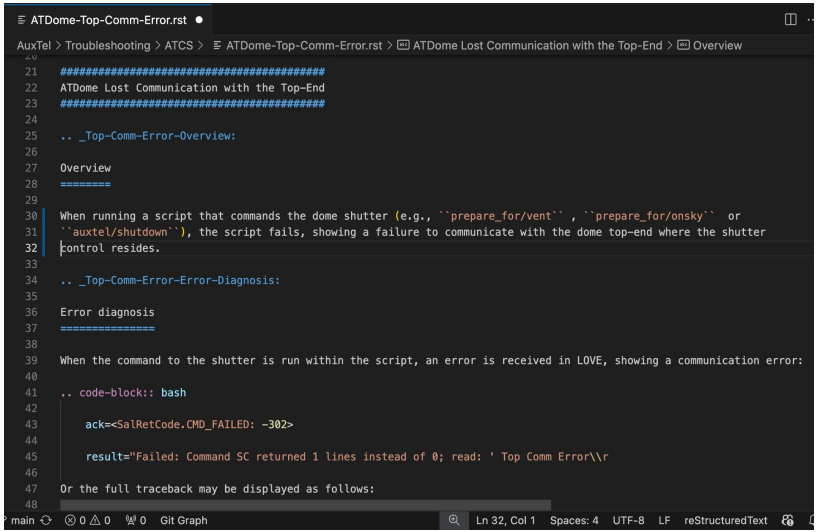
HTML File



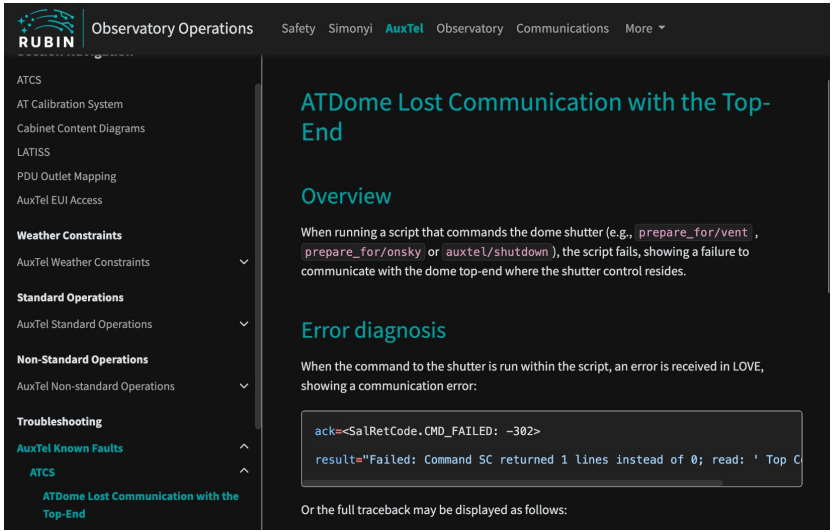
- Plaintext markup language
  - reST ⇒ Sphinx ⇒ HTML
- Examples: https://obs-ops.lsst.io/v/
- Rubin Documentation Style Guide

Editing Files – reStructuredText (reST)

reST File



HTML File



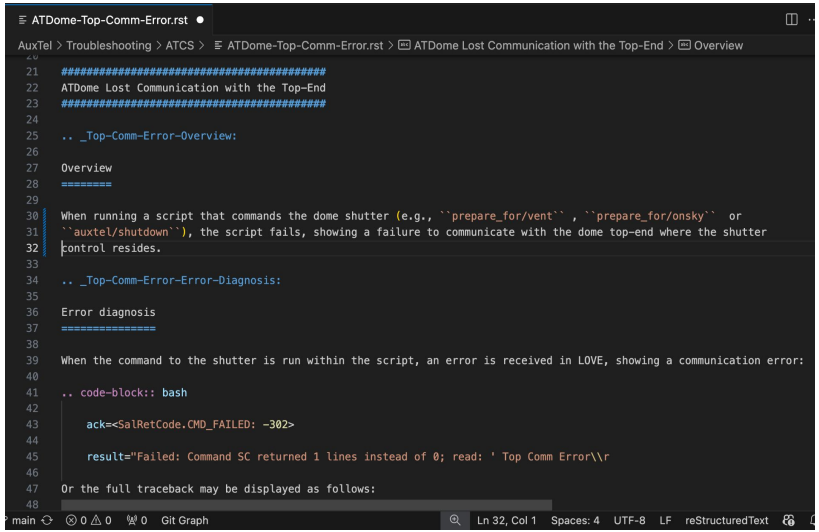
- Plaintext markup language
  - reST ⇒ Sphinx ⇒ HTML
- Examples: https://obs-ops.lsst.io/v/
- Rubin Documentation Style Guide

NOTE: Create Environment & Install Project Tools

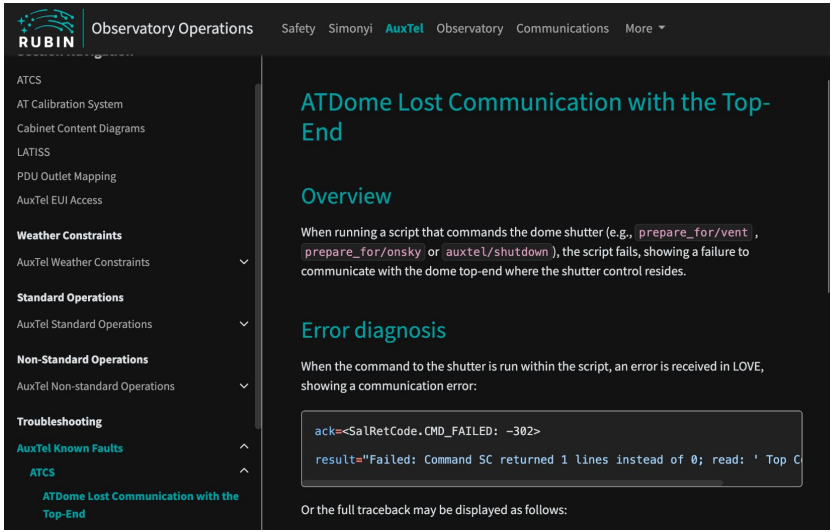
```
>> python3 -m venv .venv
>> source .venv/bin/activate

>> pip install upgrade pip
>> pip install -r requirements.txt
```

Editing Files – reStructuredText (reST)

reST File



HTML File



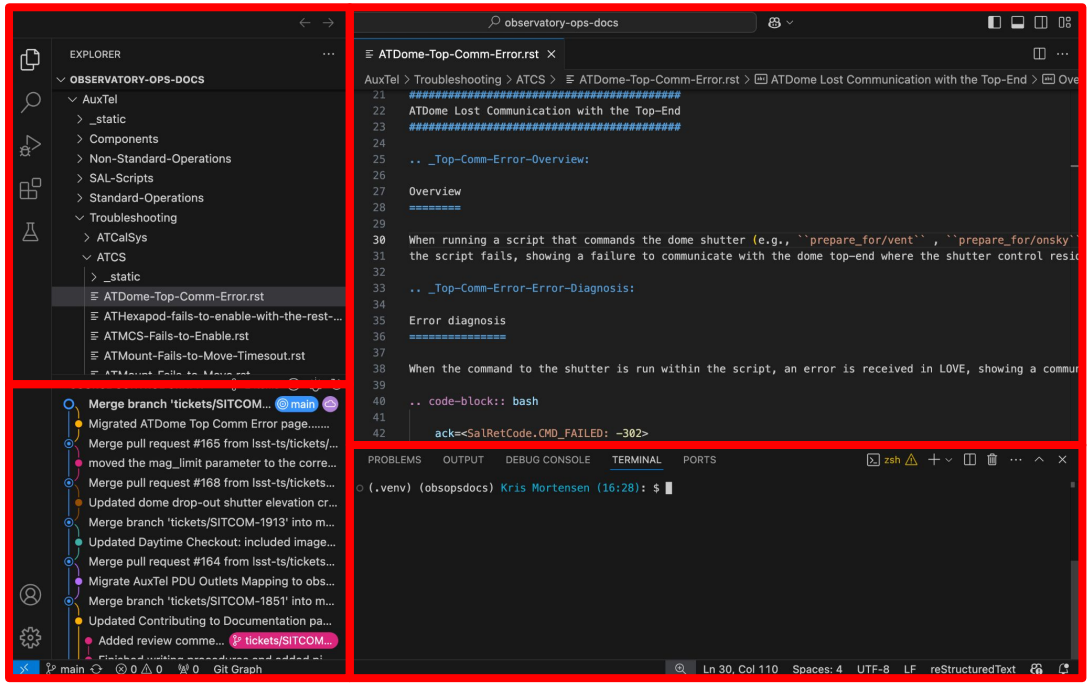
NOTE: Check reST Changes with HTML

```
>> make html
>> make linkcheck
>> open .build/html/index.html
```

NOTE: Create Environment & Install Project Tools

```
>> python3 -m venv .venv
>> source .venv/bin/activate
>> pip install upgrade pip
>> pip install -r requirements.txt
```

Editing Files – Visual Studio Code (VS Code)

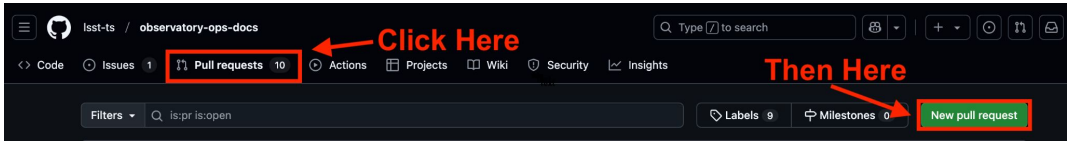


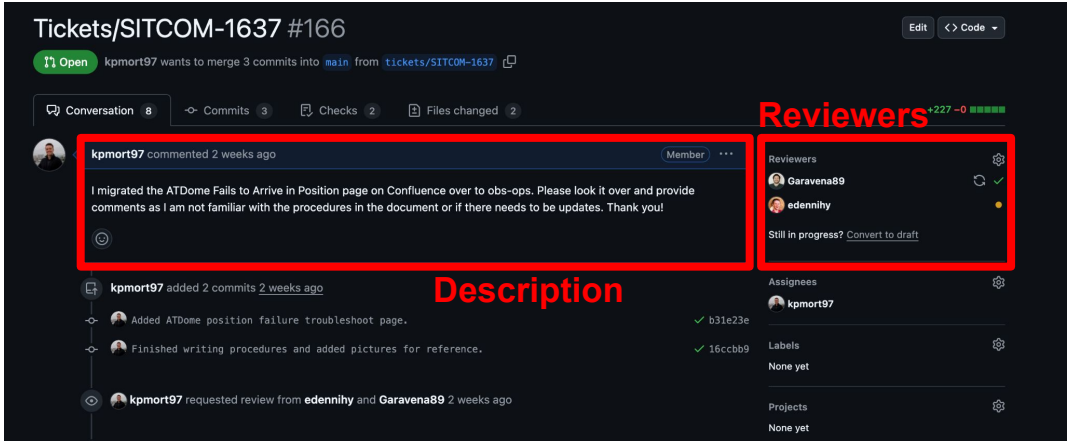
- Visual code editor
- Beneficial for any type of coding
  - Supports many programming languages
  - Syntax highlighting
  - Git integration (with commit debugging)
  - Customizable layouts to reduce clutter

- Useful Add-ons for documentation:
  - reStructuredText
  - reStructuredText Syntax Highlighting
  - Git Graph

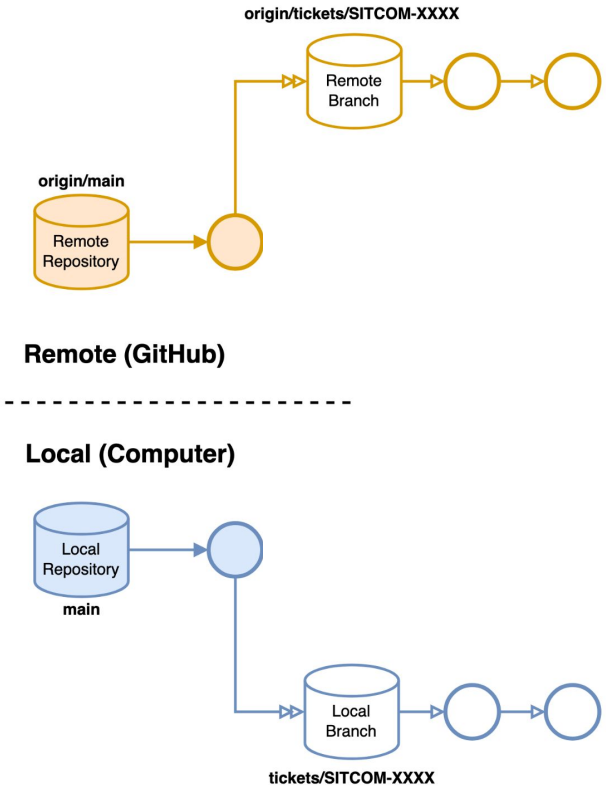
Merging a Branch – Pull Request

- Proposal to merge a set of changes from one branch into another.
  - Created in GitHub
  - Collaborators review/discuss changes before integration.
- Pull requests connect to Jira Tickets (e.g., ***Tickets/SITCOM–1637***)
- Make sure to include a description of the changes in documentation and add reviewers originally on the ticket.

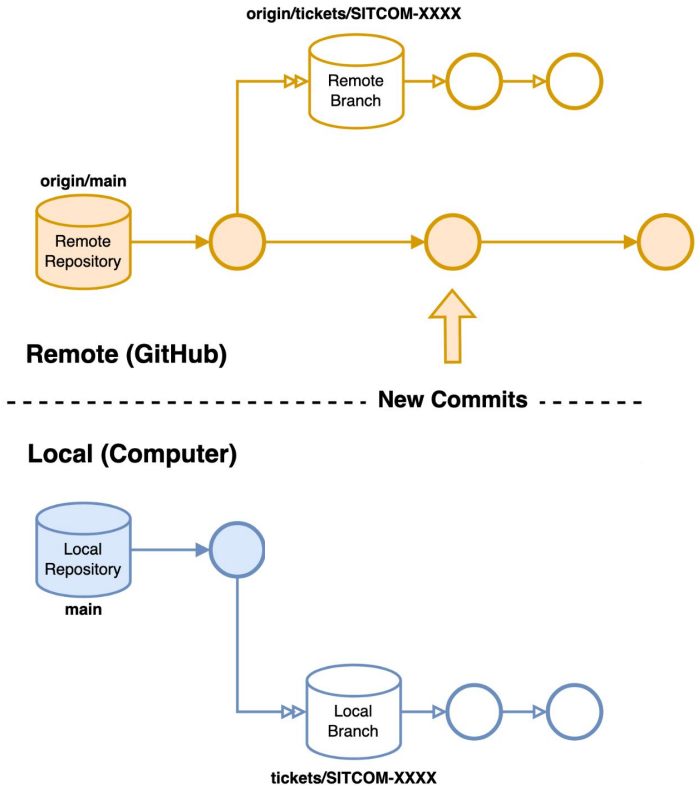




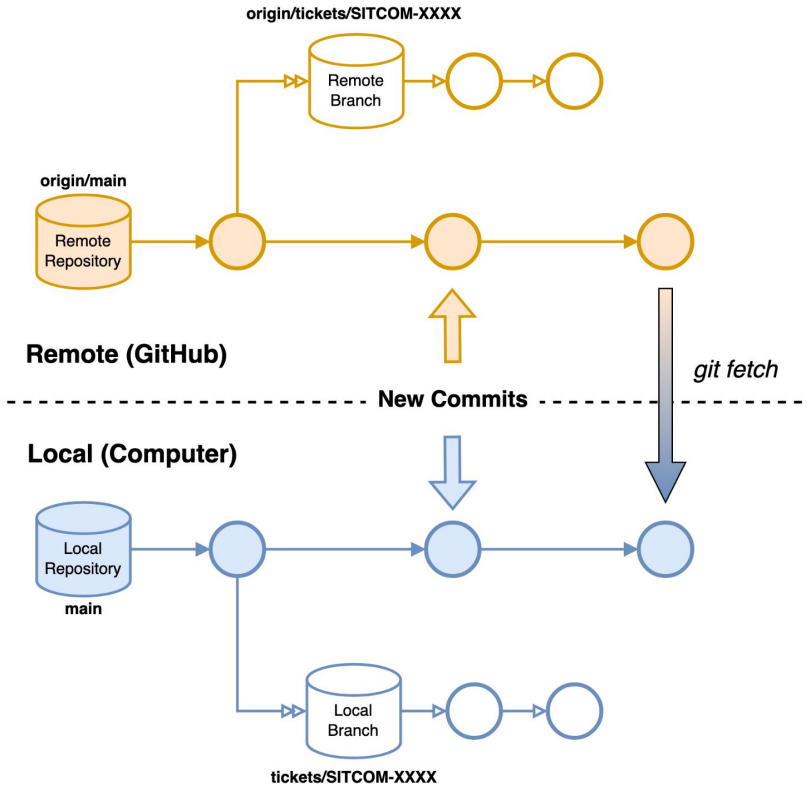
Merging a Branch – Rebasing



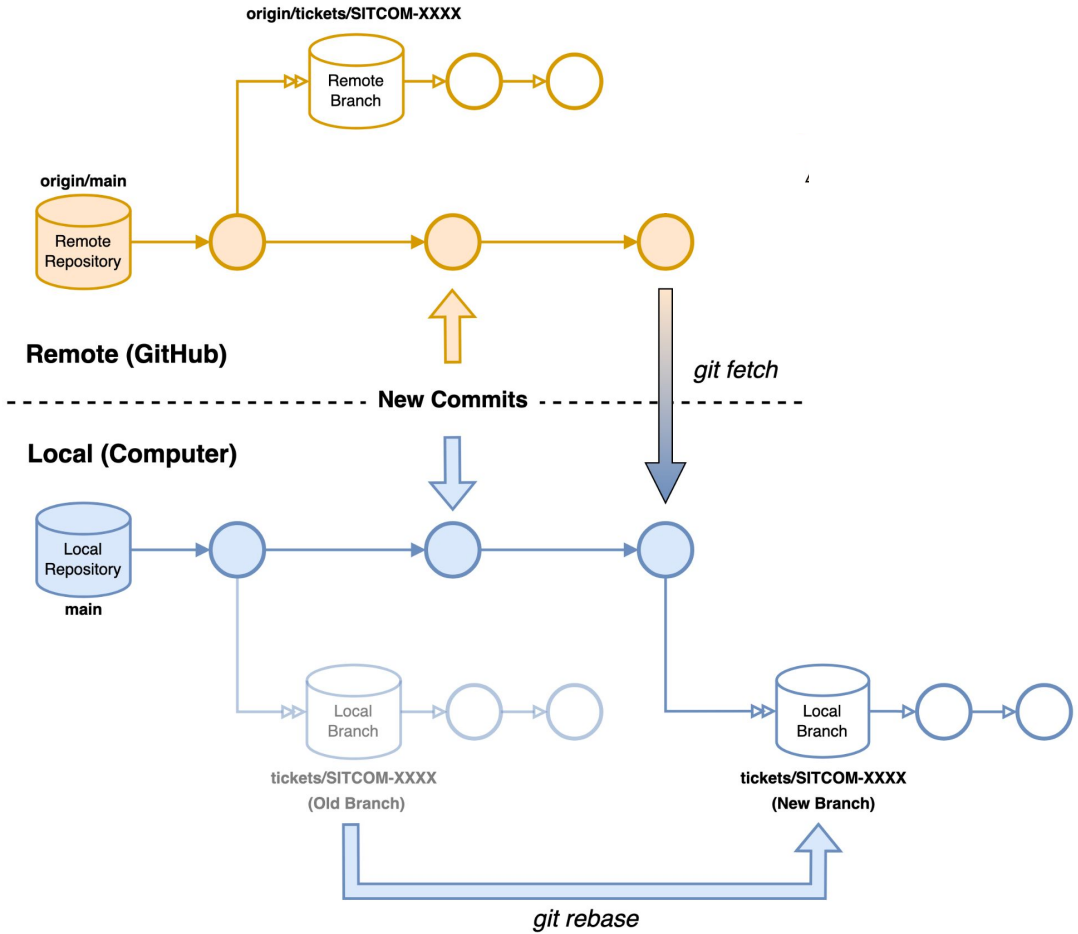
Merging a Branch – Rebasing



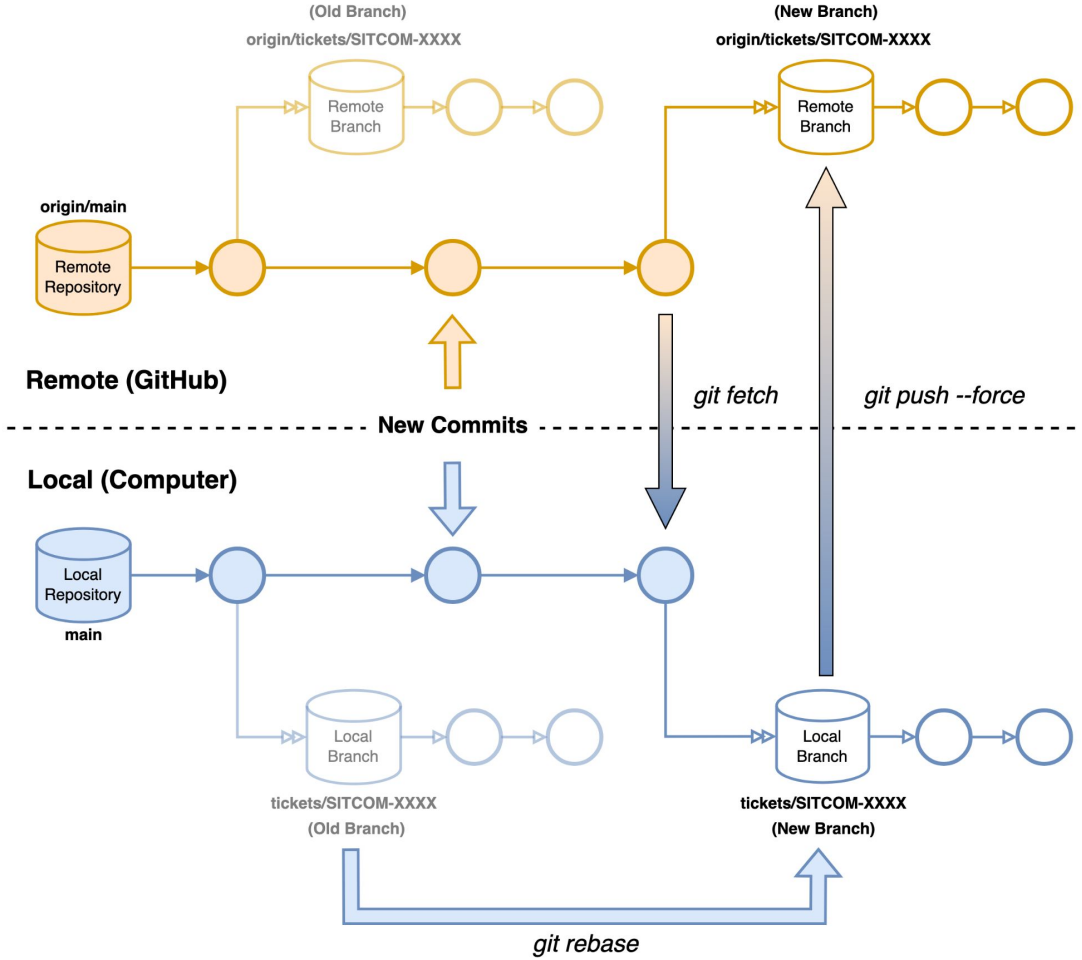
Merging a Branch – Rebasing



Merging a Branch – Rebasing

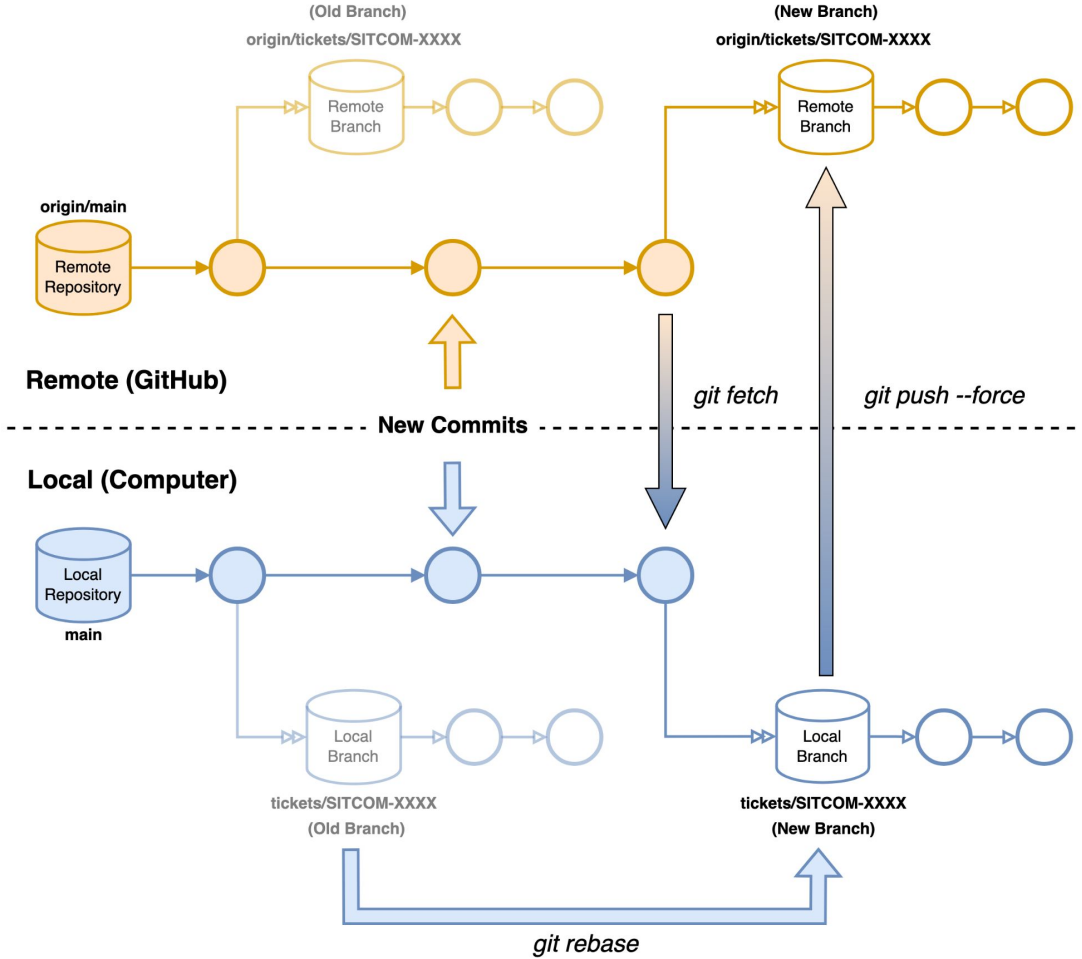


Merging a Branch – Rebasing



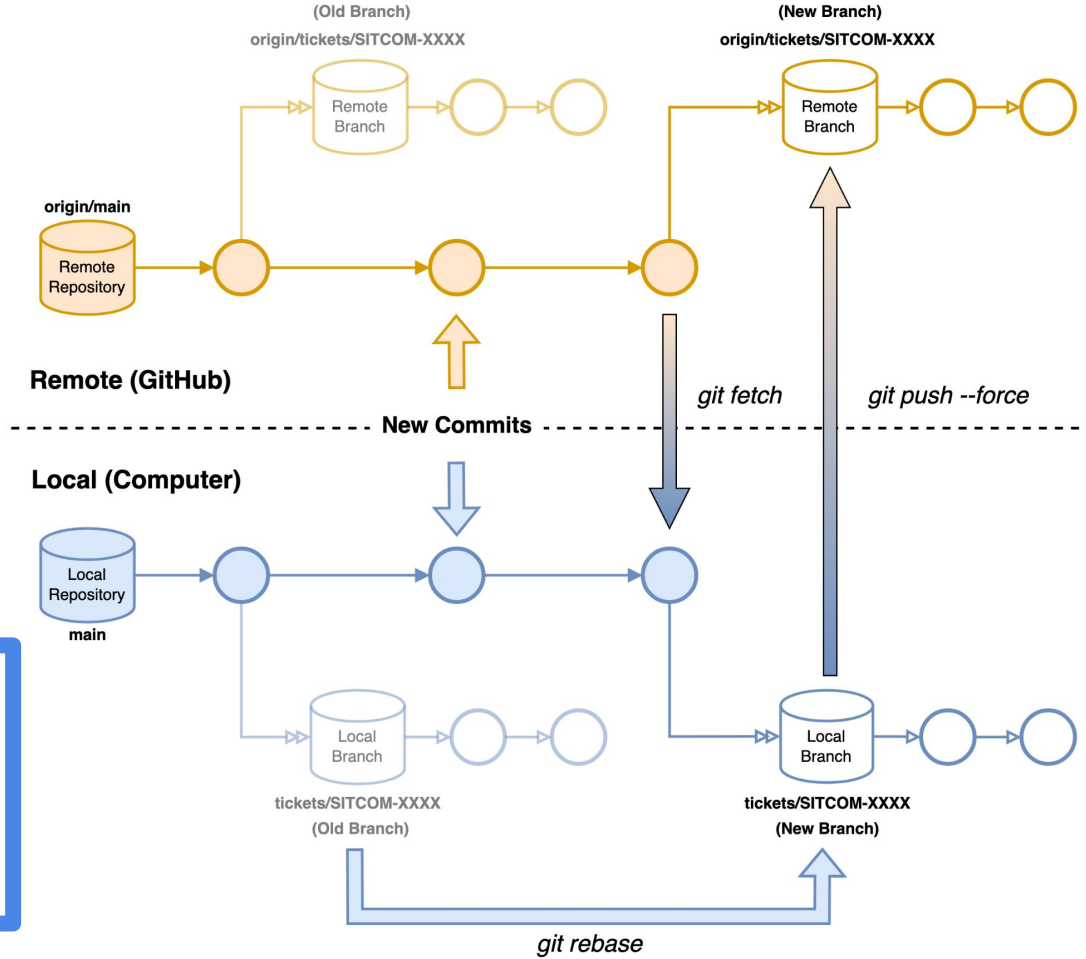
Merging a Branch – Rebasing

- Why use **`git rebase`**?
  - Streamlines (possibly) complex history.
  - Cleans up commits into a single commit, making life easier for DevOps.
  - Avoids the merge commit noise.
  - Creates a streamlined, linear log.
  - Provides a lack of clutter, allowing project movement.



Merging a Branch – Rebasing

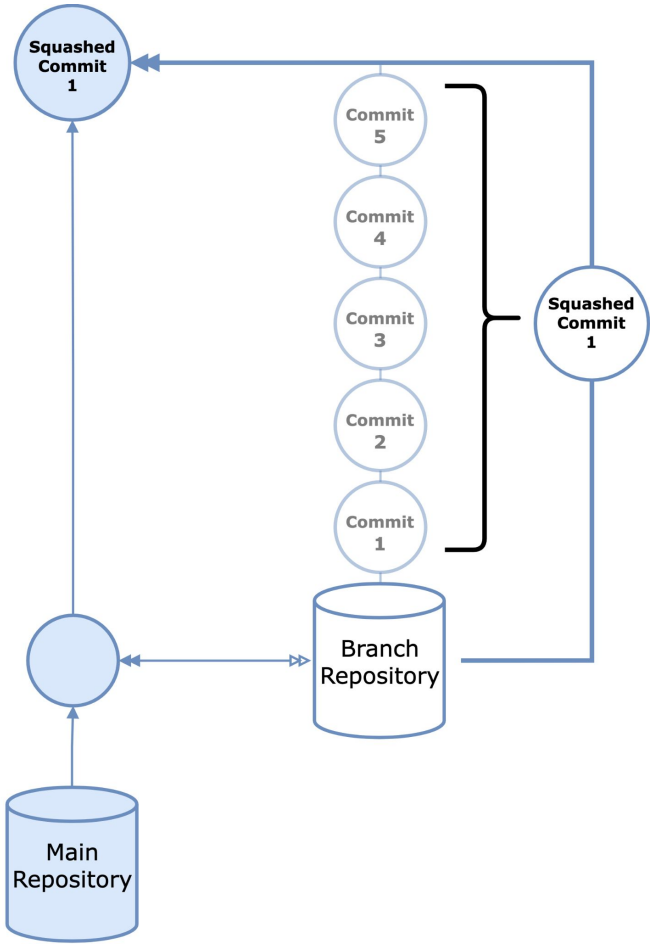
- Why use **`git rebase`**?
  - Streamlines (possibly) complex history.
  - Cleans up commits into a single commit, making life easier for DevOps.
  - Avoids the merge commit noise.
  - Creates a streamlined, linear log.
  - Provides a lack of clutter, allowing project movement.



NOTE: Common Terminal Commands

```
>> git checkout <branch>
>> git rebase origin/main

>> git push --force-with-lease
```

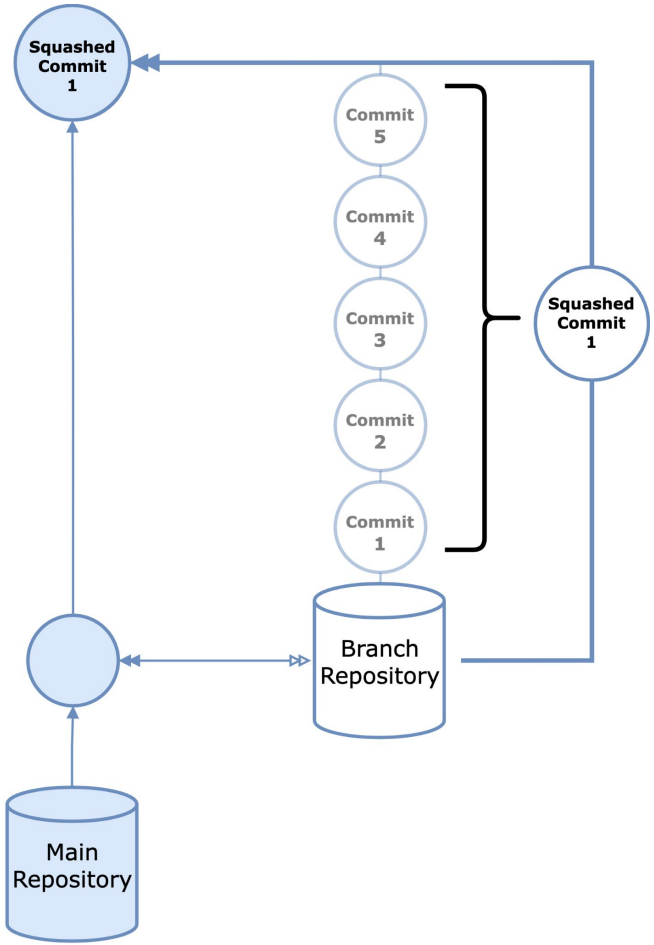
Merging a Branch – Squashing Commits



Why squash commits?

⇒ ***REDUCE CLUTTER***

Merging a Branch – Squashing Commits



Why squash commits?

⇒ ***REDUCE CLUTTER***

Two ways to squash commits:

Merging a Branch – Squashing Commits

`git rebase -i HEAD~2`

```
pick 6234378 [1759361] reST reference
pick 96ecf37 AuxTel EUI basic access procedure which includes screenshots of individual components EUI wepages
s), and initial set-up to access the remote desktop.

# Rebase 750a73a..96ecf37 onto 750a73a (2 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
```

```
# This is a combination of 2 commits.
# This is the 1st commit message:

AuxTel EUI basic access procedure which includes screenshots of individual components EUI wepages

# This is the commit message #2:

# AuxTel EUI basic access procedure which includes screenshots of individual components EUI wepage
tial set-up to access the remote desktop.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun May 12 20:44:08 2024 -0400
#
# interactive rebase in progress; onto 750a73a
# Last commands done (2 commands done):
#    pick 6234378 [1759361] reST reference
#    squash 96ecf37 AuxTel EUI basic access procedure which includes screenshots of individual com
eumatics), and initial set-up to access the remote desktop.
# No commands remaining.
# You are currently rebasing branch 'tickets/SITCOM-1359' on '750a73a'.
```

Why squash commits?

⇒ ***REDUCE CLUTTER***

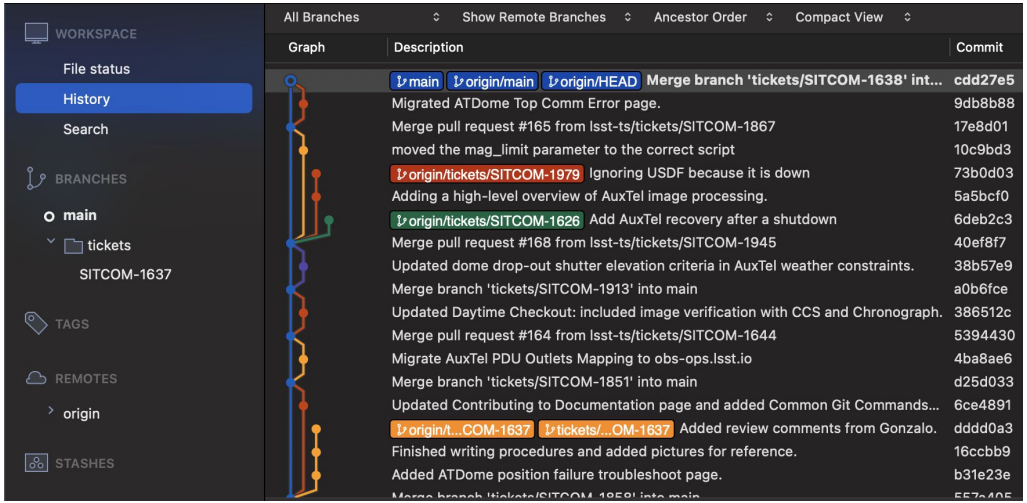
Two ways to squash commits:

(1) Git Commands & Text Editor

```
>> git rebase -i HEAD~n
>> git push --force-with-lease
```

Merging a Branch – Squashing Commits

Sourcetree – Free Git GUI



Download the program from the Sourcetree website.

Why squash commits?

⇒ ***REDUCE CLUTTER***

Two ways to squash commits:

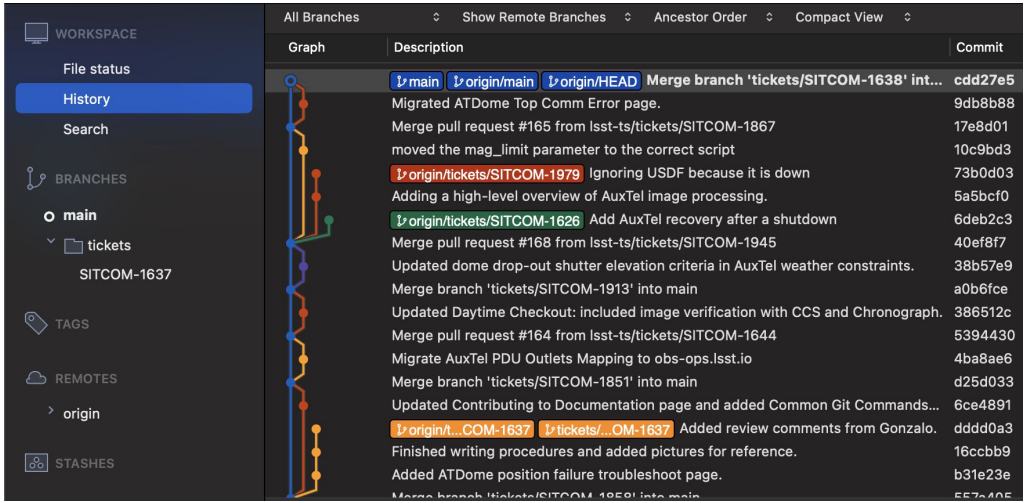
(1) Git Commands & Text Editor

```
>> git rebase -i HEAD~n
>> git push --force-with-lease
```

(2) SourceTree (Visual)

Merging a Branch – Squashing Commits

Sourcetree – Free Git GUI



Download the program from the Sourcetree website.

Why squash commits?

⇒ ***REDUCE CLUTTER***

Two ways to squash commits:

(1) Git Commands & Text Editor

```
>> git rebase -i HEAD~n
>> git push --force-with-lease
```

(2) SourceTree (Visual)

Once commits are squashed, you are ready to merge!

Committing the Final Merge

Git Commands

Update local repository.

```
>> git pull origin main
```

Switch to the main branch.

```
>> git checkout main
```

Merge ticket branch safely (preserve history).

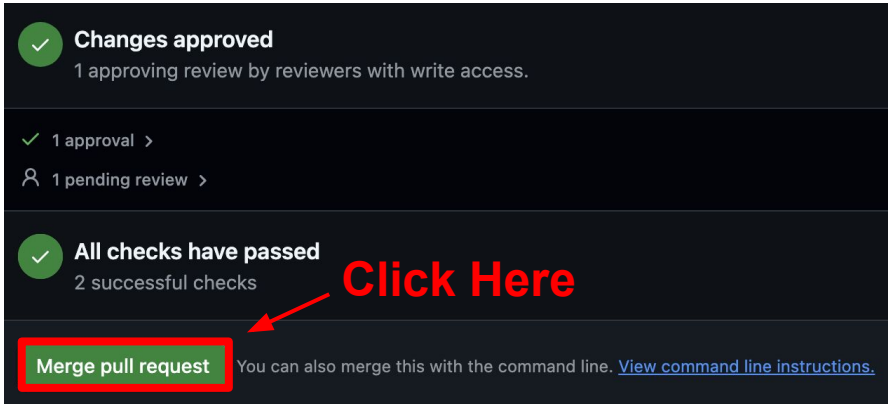
```
>> git merge <branch> --no-ff
```

Push the changes to GitHub.

```
>> git push -u origin main
```

GitHub

Within the pull request:



Summary

– Git and GitHub allow communal version control of Obs-Ops Documentation.

– There are many routes available within the documentation procedures.
  – Lots of commands, editing tools, GUIs, etc.
  – Choose the options that **work best for you**!

– Do not hesitate to look at step-by-step guides on Obs-Ops:
  – Contributing to Observatory Operations Documentation
  – Git Commands for Documentation

Extra Practice: https://github.com/kpmort97/GitHub-Tutorial-Obs-Ops

  **Reach out to Kris Mortensen for access to repository.