GitHub
GIT CHEAT SHEET

Git is the open source distributed version control system that facilitates GitHub activities on your laptop or desktop. This cheat sheet summarizes commonly used Git command line instructions for quick reference.

INSTALL GIT

GitHub provides desktop clients that include a graphical user interface for the most common repository actions and an automatically updating command line edition of Git for advanced scenarios.

GitHub for Windows
https://windows.github.com

GitHub for Mac
https://mac.github.com

Git distributions for Linux and POSIX systems are available on the official Git SCM web site.

Git for All Platforms
http://git-scm.com

CONFIGURE TOOLING

Configure user information for all local repositories

```
$ git config --global user.name "[name]"
```
Sets the name you want attached to your commit transactions

```
$ git config --global user.email "[email address]"
```
Sets the email you want attached to your commit transactions

```
$ git config --global color.ui auto
```
Enables helpful colorization of command line output

CREATE REPOSITORIES

Start a new repository or obtain one from an existing URL

```
$ git init [project-name]
```
Creates a new local repository with the specified name

```
$ git clone [url]
```
Downloads a project and its entire version history

MAKE CHANGES

Review edits and craft a commit transaction

```
$ 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 preserve its contents

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

GROUP CHANGES

Name a series of commits and combine completed efforts

```
$ 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 the working directory

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

```
$ git branch -d [branch-name]
```
Deletes the specified branch

GIT CHEAT SHEET

REFACTOR FILENAMES
Relocate and remove versioned files

```
$ git rm [file]
```

Deletes the file from the working directory and stages the deletion

```
$ git rm --cached [file]
```

Removes the file from version control but preserves the file locally

```
$ git mv [file-original] [file-renamed]
```

Changes the file name and prepares it for commit

SUPPRESS TRACKING
Exclude temporary files and paths

```
*.log
build/
temp-*
```

A text file named `.gitignore` suppresses accidental versioning of files and paths matching the specified patterns

```
$ git ls-files --other --ignored --exclude-standard
```

Lists all ignored files in this project

SAVE FRAGMENTS
Shelve and restore incomplete changes

```
$ git stash
```

Temporarily stores all modified tracked files

```
$ git stash pop
```

Restores the most recently stashed files

```
$ git stash list
```

Lists all stashed changesets

```
$ git stash drop
```

Discards the most recently stashed changeset

REVIEW HISTORY
Browse and inspect the evolution of project files

```
$ git log
```

Lists version history for the current branch

```
$ git log --follow [file]
```

Lists version history for a file, including renames

```
$ git diff [first-branch]...[second-branch]
```

Shows content differences between two branches

```
$ git show [commit]
```

Outputs metadata and content changes of the specified commit

REDO COMMITS
Erase mistakes and craft replacement history

```
$ git reset [commit]
```

Undoes all commits after `[commit]`, preserving changes locally

```
$ git reset --hard [commit]
```

Discards all history and changes back to the specified commit

SYNCHRONIZE CHANGES
Register a repository bookmark and exchange version history

```
$ git fetch [bookmark]
```

Downloads all history from the repository bookmark

```
$ git merge [bookmark]/[branch]
```

Combines bookmark's branch into current local branch

```
$ git push [alias] [branch]
```

Uploads all local branch commits to GitHub

```
$ git pull
```

Downloads bookmark history and incorporates changes

GitHub Training

Learn more about using GitHub and Git. Email the Training Team or visit our web site for learning event schedules and private class availability.

✉ **training@github.com**
training.github.com

GIT.

- open cmd
  - cd C:

Should be on C:\Users\jskoglun\Documents\ats_programming>

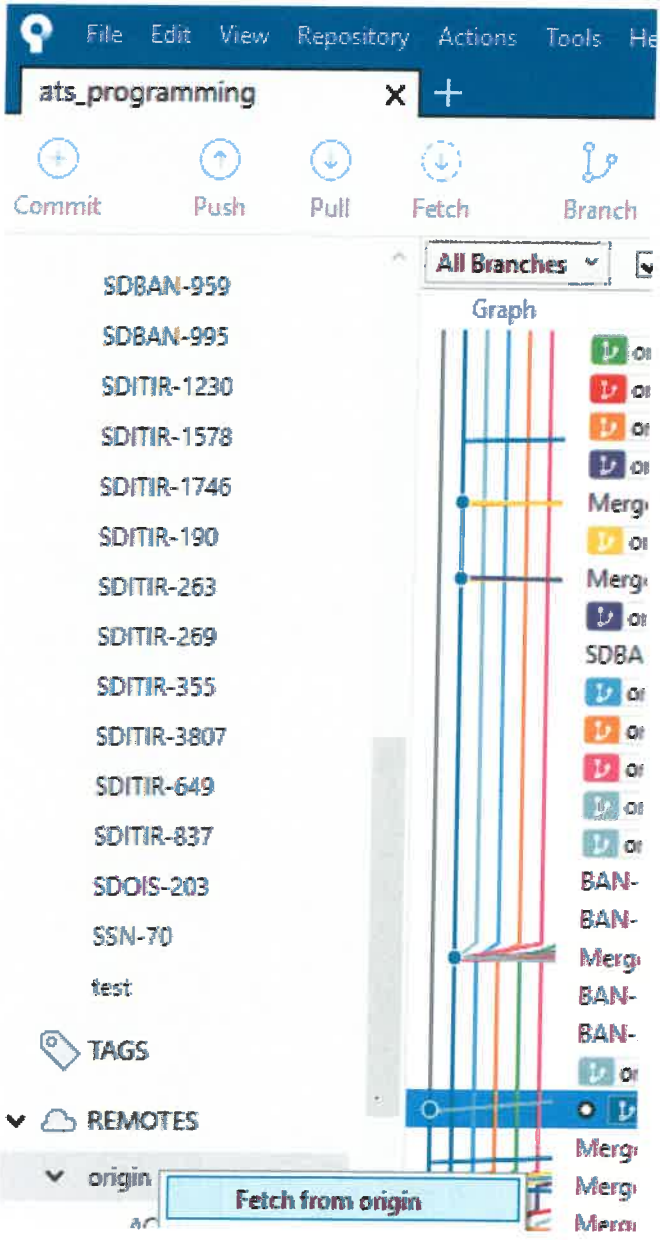
- To create a new branch:
  git checkout -b SDBAN-1004
- Locate the new file on your computer in the folder SDBAN-1004
  - git add *SDBAN*
  - git status
  - git add *Z16A* Z16A is the pkg
  - git commit -m "SDBAN-1004 updated because"
  - git push origin SDBAN-1004
  - Look in Stash in SDBAN-1004 there will be your commit
  - In Stash click on Pull Requests
    Create pull request
    Select branch SDBAN-1004
    select TO TEST
    continue add comment submit

Since the programmers have already done the set up steps for new work in Git by the time you're reviewing their work, your workflow is different from the standard steps. The high level steps, and what's covered in the document are:

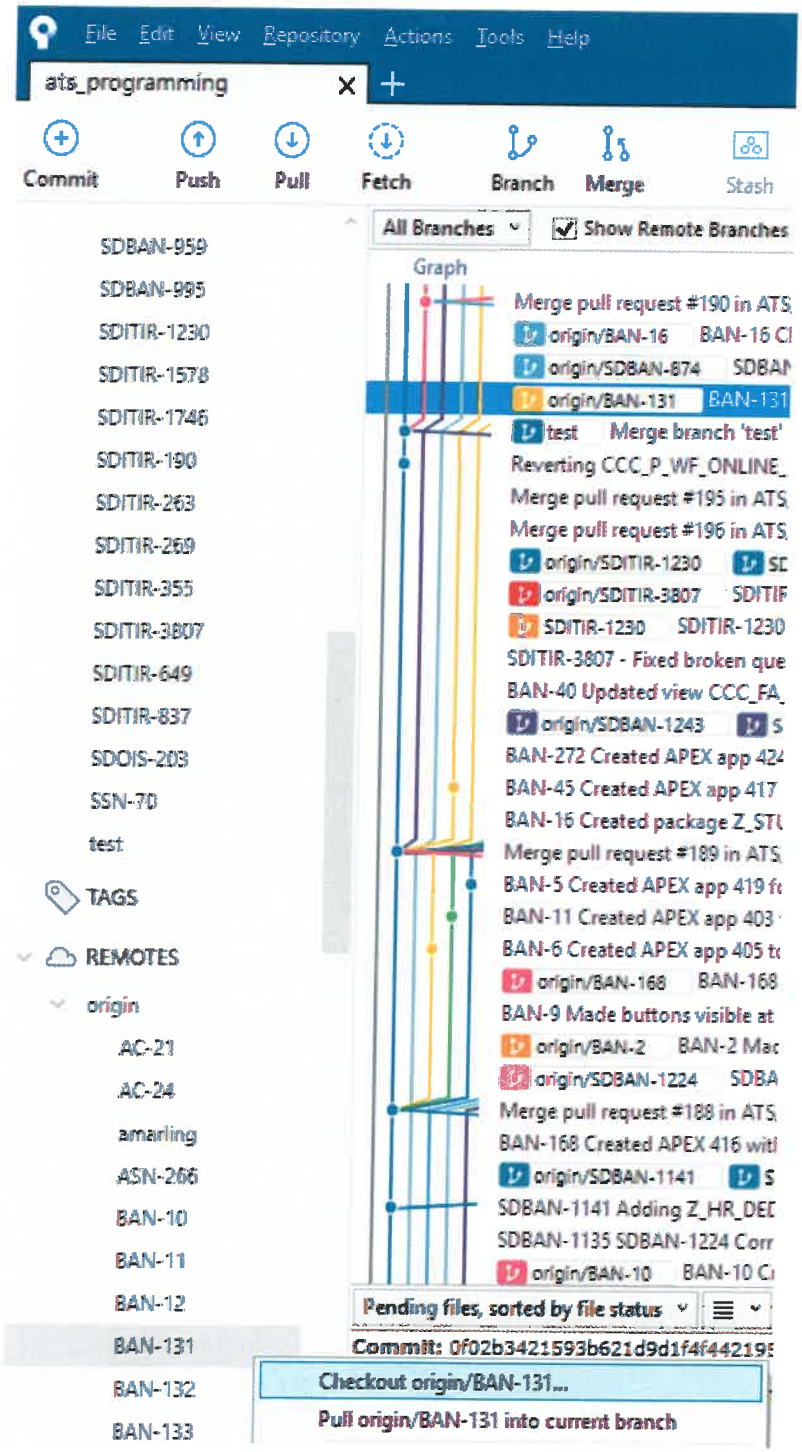
1. Fetch remote branches that the programmers have created
2. Do your work in your local repository (this step hasn't changed from the regular workflow)
3. Clean up commits if necessary and push changes in your local branches to BitBucket

Fetching remote branches:

1. Fetch the latest list of remote branches from BitBucket by right clicking the "origin" remote and clicking fetch from origin:



2. Find the branch that corresponds with the ticket you're doing an internal review on. Right click it, and check it out into your local repository (Shortcut for ***git checkout -b BRANCH --track REMOTE/BRANCH***):

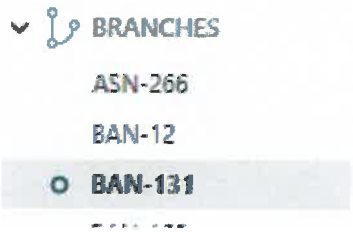


3. You're now working with your local repository. Scroll up to the "branches" section of the repository browser to continue working.

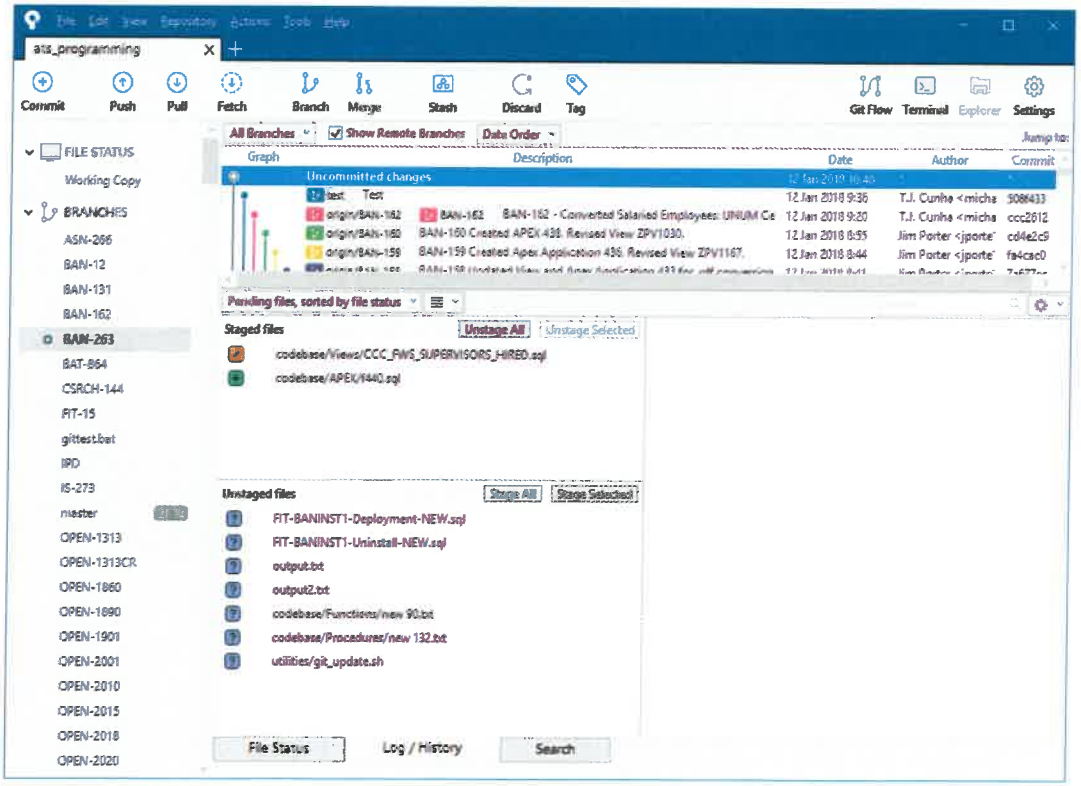
Working in your local repository:

This is where the bulk of your source control work will happen. These steps can happen in any order, and you'll need to repeat them depending on what you're trying to do.

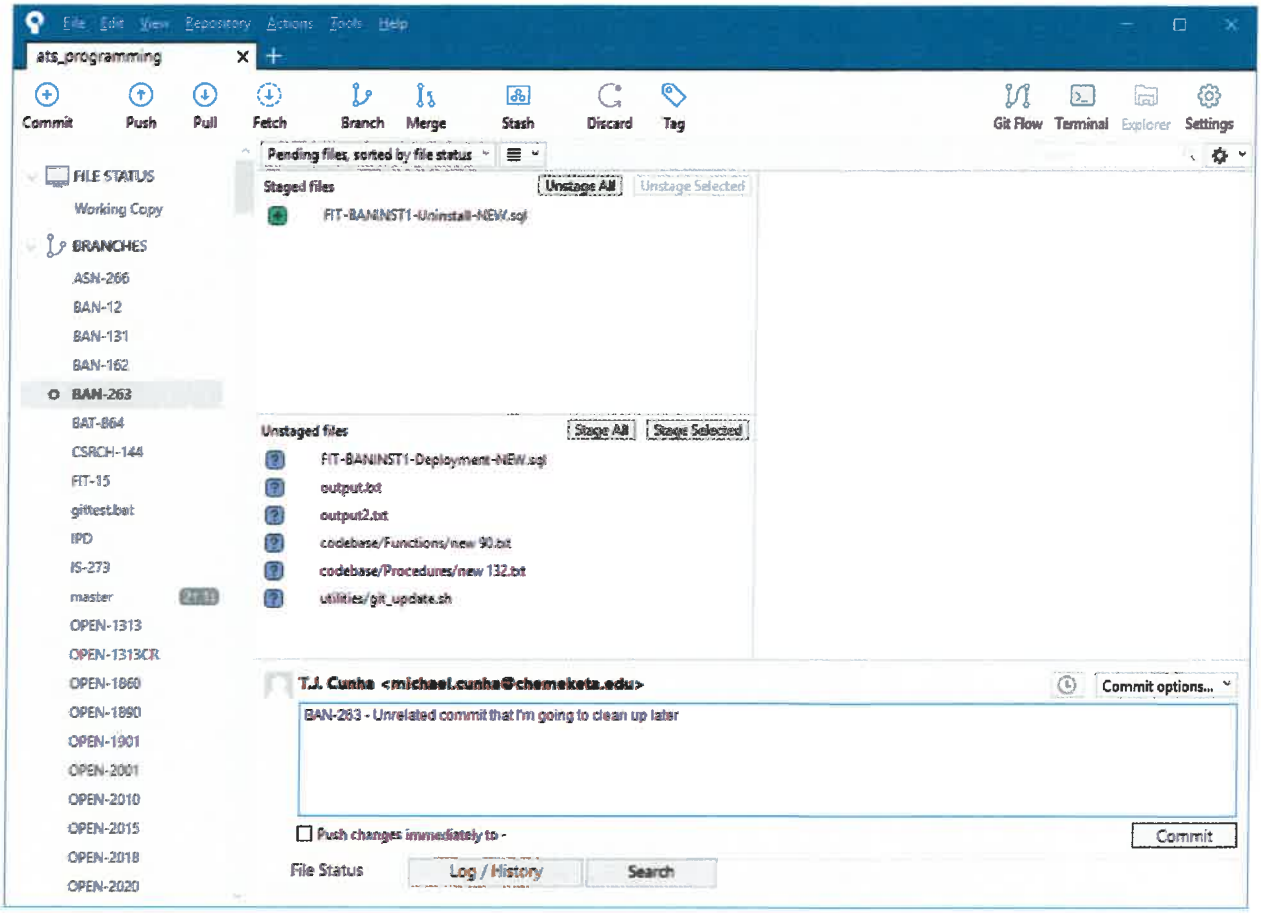
1. When you're ready to write your Internal Review changes to source control, or if you're doing work directly in the code base, double click the local branch to check it out. You can tell that a branch is checked out if appears in bold under Branches



   You want to do this before you write, replace, or modify any files in your local repository. If you miss this step, it's not a big deal, but it will create extra work later and make things confusing.
2. Drag and drop your files to the appropriate place in the "codebase" directory. If it's something new, make sure the filename follows the code base naming standard. If it already exists, replace it with your version. No screenshots because you do this in Windows Explorer.
3. Back in source tree, stage the files that have been modified (equivalent to git add). If you don't see the staging area, scroll to the top of the commit window and change your pointer to "Uncommitted changes", or open the commit menu by clicking commit on the top left:



4. Fill out your commit message, and create a new commit. If you don't see this option, click commit on the top left.

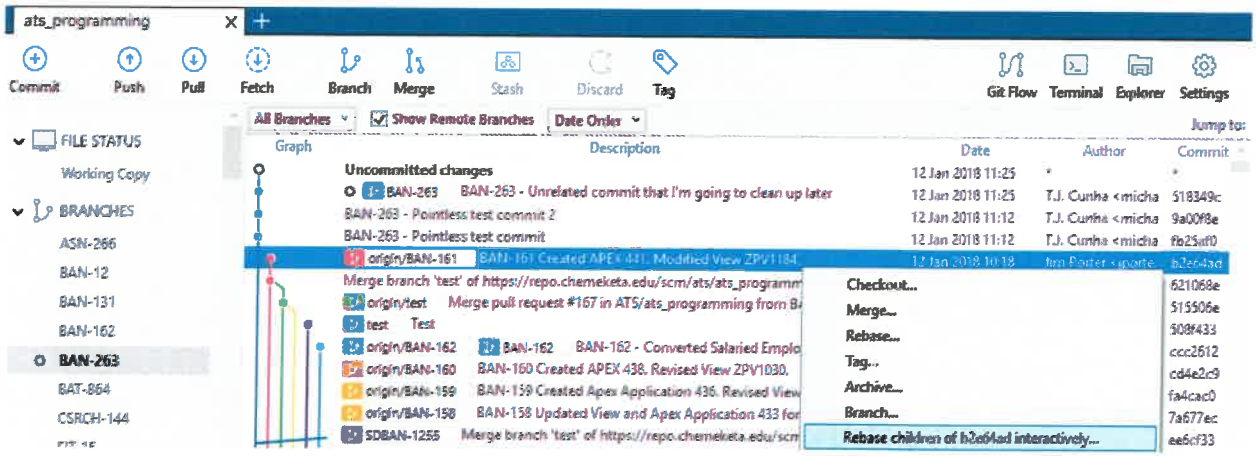


Cleaning up commits:

If you collaborate with another person on the same remote branch, commits that are written to the remote branch are final. For this reason, you might want to clean up dirty commits (partial commits, save points that don't correspond with versions, bad commit messages) before you do a push.

The most robust option here is an interactive rebase. To do an interactive rebase in Source Tree:

1. Find the commit that was made before the earliest one that you want to go back and edit. Right click this commit, and select "Rebase children of HASH interactively".



   In the screenshot above, I want to edit all of my BAN-263 commits, so I'm selecting a commit made before I made my BAN-263 commits
2. This takes me to the interactive rebase menu in Source Tree. To learn how to do an interactive rebase, refer to: https://www.atlassian.com/blog/sourcetree/interactive-rebase-sourcetree
3. Note: You can use an interactive rebase to fix bad commit messages by clicking "Amend Commit"

If you messed up a commit, and caught yourself in the moment, you can use git reset --soft HEAD~1 and git commit amend among other options to go back. The quicker you catch an issue, the easier it is to fix.

Another thing to note. If you can't figure out how to do something in Source Tree, and want to use the command line instead, you can open up git bash by clicking terminal on the upper right hand side

Command line reference (see me for more details if needed)
Checking out remote branches using the command line (Fill in the Branch name)

git checkout -b BRANCH --track origin/BRANCH

Switch between branches in your local repository

git checkout BRANCH

Create a new commit:

git status (displays what files have changed)
*git add FILES (you can use a wildcard, *, if you want)*
git commit -m "TICKET - What you changed/something to identify the commit"

Example commit messages:

"BAN-131 - Fixed formatting in APEX 437acas"
"BAN-131 - Altered where clause to include SLOR employees"

Best practice is to briefly describe what you changed since the last commit. If you don't want to though, at least put in what state you're committing it in, like "Internal Review" or "Internal Review changes" or "UAT changes"

Clean up commits before a push:

There are several ways to do this. The most powerful way is an interactive rebase.
git rebase -i COMMIT_HASH (get COMMIT_HASH from git log)
git commit --amend
git reset --soft HEAD~1