Olu Afekare

olu.afekare@reedbusiness.com

Installation and Use of the Git Bash

Git Bash

Command Line

Contents

[**Introduction** 2](#_Toc509910378)

[**What is Git Bash** 2](#_Toc509910379)

[**Installing Git** 2](#_Toc509910380)

[Installing on Linux 2](#_Toc509910381)

[Installing on Windows 2](#_Toc509910382)

[**Clone a Repository** 7](#_Toc509910383)

[**Add files to the Repository** 7](#_Toc509910384)

[Add new file in Git 7](#_Toc509910385)

[Add all new files in Git 7](#_Toc509910386)

[Save the changes in local repository 7](#_Toc509910387)

[Check branch status for changes made 7](#_Toc509910388)

[Save the changes in the remote repository 8](#_Toc509910389)

[Update my local branch from origin 8](#_Toc509910390)

[**Resolve Code merge conflicts** 9](#_Toc509910391)

[Competing line change merge conflicts 9](#_Toc509910392)

[**Resolve deleted file merge conflicts** 11](#_Toc509910393)

[**Managing a Remote** 12](#_Toc509910394)

[Add a Remote 12](#_Toc509910395)

[Rename the remote 13](#_Toc509910396)

[Remove the remote 13](#_Toc509910397)

[**Managing a Branch** 14](#_Toc509910398)

[Rename a branch 14](#_Toc509910399)

[Clone a branch 14](#_Toc509910400)

[In order to clone a particular branch, use the below command 14](#_Toc509910401)

[**View local and remote branches** 15](#_Toc509910402)

[**Add an existing project to GitHub** 16](#_Toc509910403)

[**Additional git commands** 17](#_Toc509910404)

[ Use git tag to mark a known point. 17](#_Toc509910405)

[ Use git-fetch to download objects and refs from another repository 17](#_Toc509910406)

[ Use git help to display help commands as shown below 17](#_Toc509910407)

**Introduction**

This document assumes you are familiar with Atlassian SourceTree tool (*basic knowledge around create a branch, commit, pull or push branches*) and have little or no knowledge of Git Bash. This document focuses on the use the command line tool - Git Bash. It provides vital information around how to install Git Bash, how to access it, what git command to use in a given Development situation etc.

**What is Git Bash**

Git Bash (***B****ourne* ***A****gain* ***Sh****ell*) is a command shell on Linux, Apple’s macOS and the Git for Windows. Git Bash is one of the many text-only command interfaces (CLI); it has features which allow automated scripts to be run. The installation will add a shortcut to Git Bash in the Start menu. Other shell environments will work, but are not covered in this document.

There are many command-line tools and graphical user interfaces of varying capabilities. This documentation will focus on the use of the command line tool - Git Bash. The command line is the only place you can run all Git commands unlike most GUIs, where only a partial subset of Git functionality is implemented for simplicity.

You will need to download Git Bash to your PC before you can use it. Git is quite excellent at preserving backwards compatibility, any version after 2.0.0 should be fine. However, it is better to have the latest version.

**Installing Git**

You can either install Git tools as a package or via another installer, or download the source code and compile it.

### Installing on Linux

If you want to install the basic Git tools on Linux via a binary installer, you can generally do so through the basic package-management tool that comes with your distribution. You can use *dnf*:

|  |
| --- |
| $ sudo dnf install git-all |

If you are on a Debian-based distribution like Ubuntu, use *apt-get*:

|  |
| --- |
| $ sudo apt-get install git-all |

### Installing on Windows

To install Git Bash on windows, click on the applicable link below to download the installation files:

#### Git for Windows Setup

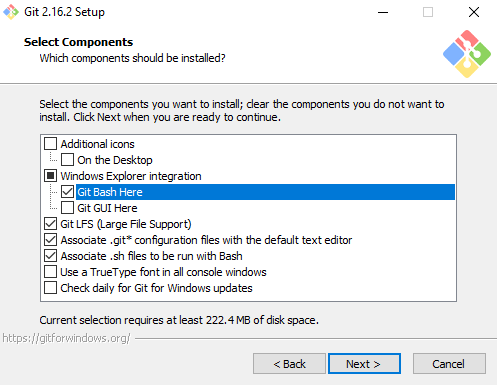
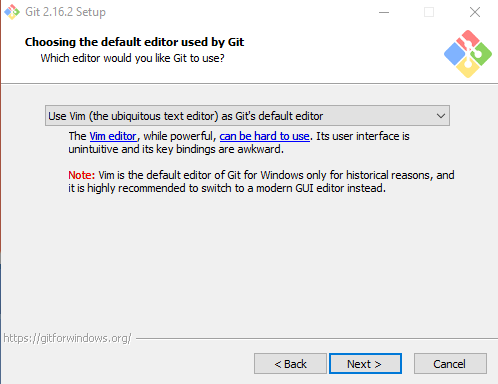
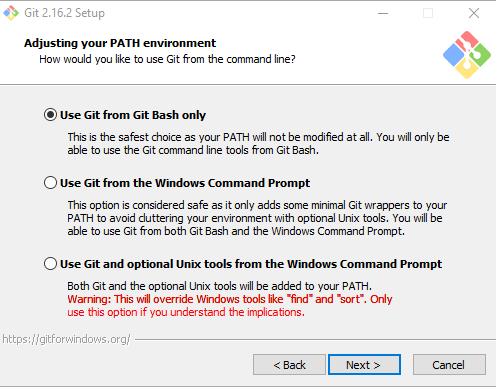
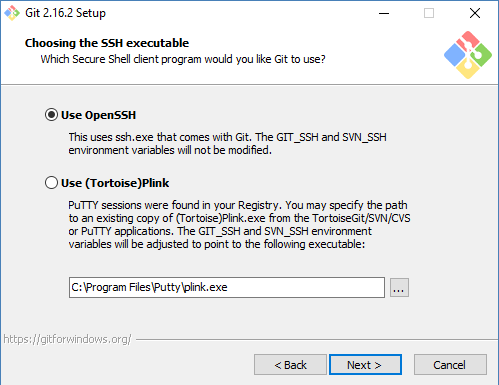
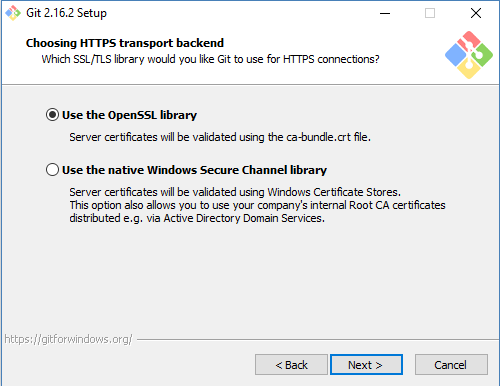
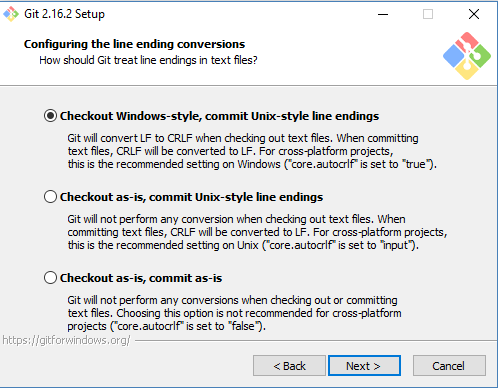
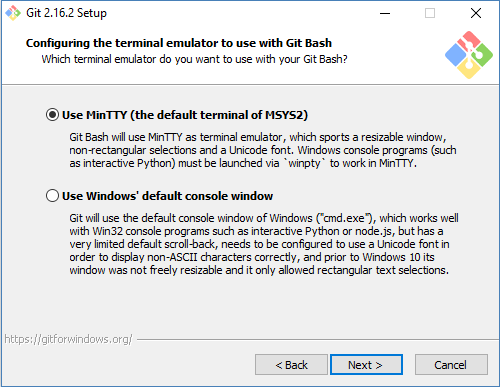
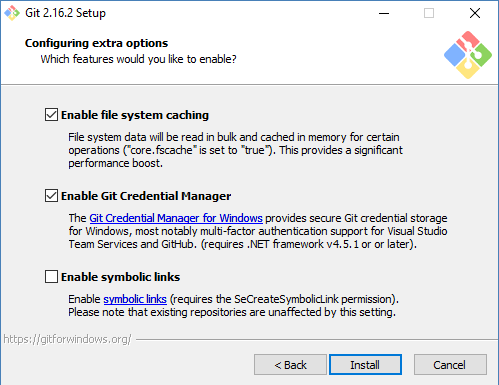
[**32-bit Git for Windows Setup**](https://github.com/git-for-windows/git/releases/download/v2.16.2.windows.1/Git-2.16.2-32-bit.exe)

[**64-bit Git for Windows Setup**](https://github.com/git-for-windows/git/releases/download/v2.16.2.windows.1/Git-2.16.2-64-bit.exe)

#### Git for Windows Portable Device (enables computers to communicate with attached media and storage devices)

[**32-bit Git for Windows Portable**](https://github.com/git-for-windows/git/releases/download/v2.16.2.windows.1/PortableGit-2.16.2-32-bit.7z.exe)

[**64-bit Git for Windows Portable**](https://github.com/git-for-windows/git/releases/download/v2.16.2.windows.1/PortableGit-2.16.2-64-bit.7z.exe)

1. During the installation, ensure Git Bash is selected: 
2. You have the options to accept the default terminal based text editor (Vim) or select a different GUI based text editor.  
    
3. Select Use Git from Git Bash only  
   
4. Select OpenSSH (SSH keys connect you to GitHub without supplying your username or password at each visit)  
   
5. Select Use the OpenSSL library  
   
6. Select Checkout Windows-style, commit Unix-style line endings  
   git stat
7. Select Use MinTTY  
   
8. Check Enable file system caching and git credential manager (*if not defaulted*)

**Clone a Repository**

At the top of your GitHub repository, click to copy the remote repository URL. Then, use the copied URL in the below command to clone a repository

git clone https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git

Create a new branch

Use the below commands to create and check out your new branch, so you can start development tasks on this new branch.

git branch\_name

git checkout branch\_name

**Add files to the Repository**

git Add new file in Git

You can either specify a filename or file path and git will add all changed files in that directory/path

git add filename

git add /path/documents/location/OA

Add all new files in Git

The git add . will adds all modified and new (untracked) files in the current directory and all subdirectories to the staging area, hence preparing them to be included in your first or next **git** commit.

git add .

# Adds the files in the local repository and stages them for commit. To unstage a file, use 'git reset HEAD Filename.

Save the changes in local repository

After making all required changes to your file(s), you need to commit the changes to your branch. Use the below command to save the changes in local repository

git commit -m "Just changes files for ADB-1111s"

Check branch status for changes made

The below command will list all current files and the current status if the files are staged. You can see all changes to your local repository and the current status of staged/committed files.

git status

Save the changes in the remote repository

Once you have committed your changes, you will need to save them remotely. Use the below command to push the changes in your local branch to the remote repository. Please note that if this is a new branch, then you will need to set the upstream Git push by using Git push –set-upstream origin (ADB-185\_Update\_Transaction\_to\_include\_USD\_and\_EUR\_values)**.**

The git push command takes two arguments: A remote name, for example, **origin**; a branch name, e.g. **ADB-185\_Update\_Transaction\_to\_include\_USD\_and\_EUR\_values**.

git push origin (ADB-185\_Update\_Transaction\_to\_include\_USD\_and\_EUR\_values)

Update my local branch from origin

Use the below command to update the local repository with changes in the remote

git pull

Merge Request

**Feature Branch(es) to Story Branch**

To merge your feature branch(es) to the Story branch, you will need to create a Merge Request on GitLab. This is assuming you have access to GitLab (<https://gitlab.coa.rbxd.git> pulls/RBICSG/ScoutReplacement/) and assign it to a co-Developer to be code reviewed. Then the co-developer will merge your branch into the Story branch if everything is correct

**Story Branch (es) to Development Branch**

To merge your Story branch to the Development branch, you will need to create a Merge Request on GitLab (<https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/>) and assign it to a co-Developer to be code reviewed. Then the co-developer will merge your branch into the Story branch if everything is correct

**Development Branch to Story Branch**

There are times when you will need to merge the Development (*parent*) branch into your Story branch to update your branch with any new changes in the Development branch. **Please note that Developers are no longer allowed to create a Merge Request** in Gitlab as this can lead to unexpected changes on the branches. See the section of this document (*Resolve code merge conflicts*) on how to resolve the merge conflicts.

To use the command line, you will need to check out the branch you want to merge the changes into, in this case, your branch.

afekareo@QHSW30702 MINGW64 ~/documents/repohpcc/joesteeleperkinstest (Olu\_test3)

$ git merge Olu\_Development

Updating 338b1d4..dae84e7

Fast-forward

Account.ecl | 9 +++++++++

1 file changed, 9 insertions(+)

create mode 100644 Account.ecl

**Resolve Code merge conflicts**

A merge conflict needs to be resolved before a feature branch can be merged into a story branch or a story branch to the Development branch. If you have a merge conflict on the command line, you cannot push your local changes to GitHub until you [resolve the merge conflict locally on your computer](https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/).

If you try merging branches on the command line that have a merge conflict, you'll get an error message like the below example:

|  |
| --- |
| git merge *BRANCH-JSP*  Auto-merging scoutreplaceEx.md  CONFLICT (content): Merge conflict in scoutreplaceEx.md  Automatic merge failed; fix conflicts and then commit the result |

You can resolve the [merge conflicts](https://help.github.com/articles/about-merge-conflicts) using the command line and a text editor (e.g. Notepad).

Competing line change merge conflicts

To resolve a merge conflict caused by competing line changes, you must choose which changes to incorporate from the different branches in a new commit.

For example, if you and another Developer edited *Account.ecl* on same line in different branches of the same Git repository, you'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

1. Open Git Bash.
2. Navigate to your local Git repository that has the merge conflict.

cd *REPOSITORY-NAME*

1. Generate a list of the files affected by the merge conflict. In this example, the file *Account.ecl* has a merge conflict. Use git status to see the conflict(s).

afekareo@QHSW30702 MINGW64 ~/documents/repohpcc/joesteeleperkinstest (OLU\_Test3)

$ git status

On branch OLU\_Test3

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

new file: Account.ecl

Changes not staged for commit:

(use "git add <file>..." to update what will be committed)

(use "git checkout -- <file>..." to discard changes in working directory)

modified: Account.ecl

If you’re 100% certain of the changes you should accept, then use git add filename (in this example, use git add Account.ecl) to commit the changes.

Then, use git commit –m to commit your changes as shown below

afekareo@QHSW30702 MINGW64 ~/documents/repohpcc/joesteeleperkinstest (Olu\_test1)

$ git commit -m "Resolved merge conflict by accepting the changes"

[Olu\_test1 dae84e7] Resolved merge conflict by accepting the changes

1 file changed, 9 insertions(+)

create mode 100644 Account.ecl

afekareo@QHSW30702 MINGW64 ~/documents/repohpcc/joesteeleperkinstest (Olu\_test1)

$

1. Alternatively, open a text editor and navigate to the file that has the merge conflicts.
2. To see the beginning of the merge conflict in your file, search the file for the conflict marker <<<<<<<.

When you open the file in your text editor, you'll see the changes from the HEAD or base branch after the line <<<<<<< HEAD. Next, you'll see =======, which divides your changes from the changes in the other branch, followed by >>>>>>> BRANCH-NAME.

In this example, one Developer added "Processing.GenerateActiveAccountAggTran" in the base or HEAD branch and another Developer added "Processing.GenerateActiveAccountAggOutput" in the compare branch or branch-a.

If (a =’JSP’ AND b <>’JSP’,

<<<<<<< HEAD

Processing.GenerateActiveAccountAggTran

=======

Processing.GenerateActiveAccountAggOutput

>>>>>>> branch-a

The area where a pair of conflicting changes happened is marked with markers <<<<<<<, =======, and >>>>>>>. The part before the ======= is typically your side, and the part afterwards is typically their side.

1. Decide if you want to keep only your changes, keep only the other branch's changes, or make a brand new change, which may incorporate changes from both branches.
2. Delete the conflict markers <<<<<<<, =======, >>>>>>> and make the changes you want in the final merge. In this example, both changes are incorporated into the final merge:

If (a =’JSP’ AND b <>’JSP’, Processing.GenerateActiveAccountAggTran, Processing.GenerateActiveAccountAggOutput

1. Add or stage your changes.

git add filename

1. Commit your changes with a comment.

git commit -m "Resolved merge conflict by incorporating both scripts."

1. You can now merge the branches on the command line or [push your changes to your remote repository](https://help.github.com/articles/pushing-to-a-remote/) on GitHub and [merge your changes](https://help.github.com/articles/merging-a-pull-request/) in a pull request.

**Resolve deleted file merge conflicts**

To resolve a merge conflict caused by competing changes to a file, where a Developer deletes a file in one branch and another Developer edits the same file, you must choose whether to delete or keep the removed file in a new commit.

For example, if you edited a file, such as *Account.md*, and another person removed the same file in another branch in the same Git repository, you'll get a merge conflict error when you try to merge these branches. You must resolve this merge conflict with a new commit before you can merge these branches.

1. Open Git Bash.
2. Navigate into the local Git repository that has the merge conflict.

cd *Documents*

cd *RepoHPCC*

1. Generate a list of the files affected by the merge conflict. In this example, the file *Account.md* has a merge conflict.

git status

# On branch master

# Your branch and 'origin/master' have diverged,

# and have 1 and 2 different commits each, respectively.

# (use "git pull" to merge the remote branch into yours)

# You have unmerged paths.

# (fix conflicts and run "git commit")

#

# Unmerged paths:

# (use "git add/rm ..." as appropriate to mark resolution)

#

# deleted by us: Account.md

#

# no changes added to commit (use "git add filename" and/or "git commit -a")

1. Open a text editor and navigate to the file that has merge conflicts.
2. Decide if you want keep the removed file. You may want to view the latest changes made to the removed file in your text editor.

To add the removed file back to your repository:

git add filename.md

To remove this file from your repository:

git rm filename.md

filename.md: needs merge

rm filename.md'

1. Commit your changes with a comment.

git commit -m "Resolved merge conflict by keeping Account.md file."

[branch-d 6f89e49] Merge branch 'branch-c' into branch-d

You can now merge the branches on the command line or [push your changes to your remote repository](https://help.github.com/articles/pushing-to-a-remote/) on GitHub and [merge your changes](https://help.github.com/articles/merging-a-pull-request/) in a pull request.

**Managing a Remote**

Add a Remote

At the top of your GitHub repository, click to copy the remote repository URL. To add a new remote, paste the copied URL into the git remote add command, in the directory that your repository is stored at.

The git remote add command takes two arguments:

* A remote name, for example, origin
* A remote URL, for example, https://github.com/user/repo.git

git remote add origin *https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git*

# Sets the new remote

git remote -v

# Verifies the new remote URL

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

Rename the remote

Use the git remote rename command to rename an existing remote. The git remote rename command takes two arguments:

* An existing remote name, for example, origin
* A new name for the remote, for example, destination

git remote -v

# View existing remotes

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

git remote rename origin destination

# Change remote name from 'origin' to 'destination'

git remote -v

# Verify remote's new name

destination https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

destination https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

Remove the remote

Use the git remote rm command to remove a remote URL from your repository. The git remote rm command takes one argument - a remote name, for example, destination

These examples assume you're [cloning using HTTPS](https://help.github.com/articles/which-remote-url-should-i-use/#cloning-with-https-urls-recommended), which is recommended.

destination https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

destination https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

git remote -v

# View current remotes

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

destination https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

destination https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

git remote rm destination

# Remove remote

git remote -v

# Verify it's gone

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (fetch)

origin https://gitlab.coa.rbxd.ds/RBICSG/ScoutReplacement/RepoHPCC.git (push)

**Note**: git remote rm does not delete the remote repository from the server. It simply removes the remote and its references from your local repository.

**Managing a Branch**

Rename a branch

Sometimes you may need to rename a branch for one reason or the other. For instance, you could have named a branch incorrectly and pushed it to the remote repository.To rename the branch, please follow these steps before other Developers work on it:

**1. Rename your local branch.**

If you are on the branch you want to rename:

git branch -m new-name

If you are on a different branch:

git branch -m old-name new-name

**2. Delete the old-name remote branch and push the new-name local branch.**

git push origin :old-name new-name

**3. Reset the upstream branch for the new-name local branch.**  
Switch to the branch and then:

git push origin –u new-name

Clone a branch

In order to clone a particular branch, use the below command

git clone https://gitlab.coa.rbxd.ds/Steelepj/JoeSteelePerkinsTest -b Olu\_Test1

**View local and remote branches**

Sometimes you may need to figure out what [branches](http://gitready.com/beginner/2009/01/25/branching-and-merging.html) exist on a remote repository so you can pull them down and check them out, merge them into your local branches, etc.

The easiest way is to use the git branch command with various options. -a shows all local and remote branches, while –r shows only remote branches.

$ git branch

\* master

$ git branch -a

\* master

origin/1-2-stable

ADB-33\_Ability\_to\_identify\_whether\_users\_have\_active\_

ADB\_244InternalUser\_Flag\_addition\_is\_missing\_in\_AccuityIBAN

Development

Merge\_Conflicts\_Test

master

origin/2-0-stable

origin/2-1-stable

origin/2-2-stable

origin/3-0-unstable

origin/HEAD

origin/master

$ git branch -r

remotes/origin/193\_Add\_GeoRegion\_lookup\_translation\_to\_Accuity\_Usage\_ECLs

remotes/origin/197\_-\_Add\_GeoRegion\_lookup\_translation\_to\_ICIS\_Usage\_ECLs

remotes/origin/ADB-111

remote/1-2-stable

remote/2-0-stable

remote/2-1-stable

remote/2-2-stable

remote/3-0-unstable

remote/HEAD

remote/master

Therefore, once you know the name of the branch, it is quite simple to [check them out.](http://gitready.com/intermediate/2009/01/09/checkout-remote-tracked-branch.html) The branches that are not pulled down are listed in red.

There is also another way to figure out what branches are on your remote by actually using the remote related commands, git remote and git ls-remote.

The former shows a lot of information about the remote in general and how it relates to your own repository, while the latter simply lists all references to branches and tags that it knows about.

$ git remote show origin

\* remote origin

URL: git://github.com/rails/rails.git

Remote branch merged with 'git pull' while on branch master

master

Tracked remote branches

1-2-stable 2-0-stable 2-1-stable 2-2-stable 3-0-unstable master

$ git ls-remote --heads origin

5b3f7563ae1b4a7160fda7fe34240d40c5777dcd refs/heads/1-2-stable

71926912a127da29530520d435c83c48778ac2b2 refs/heads/2-0-stable

2b158543247a150e8ec568becf360e7376f8ab84 refs/heads/2-1-stable

b0792a3e7be88e3060af19bab01cd3d26d347e4c refs/heads/2-2-stable

d6b9f8410c990b3d68d1970f1461a1d385d098d7 refs/heads/3-0-unstable

f04346d8b999476113d5e5a30661e07899e3ff80 refs/heads/master

ac7a7e075619f9555b1c78c4720db3f5613055e4 refs/heads/SR1065-Additional\_Changes

da0e37a5770ed4eeee375e70ca00b1a8d496de24 refs/heads/SR1067\_SessionCalculation\_UsageData

9ddcd7b66fe2b991c14d8f76f9949ddfade3175e refs/heads/SR1147

a80cef8b027013409c90fa838f27948d3aef7f4a refs/heads/SR1212-Additional\_Changes

805d14600ef05aed06bb6aeac18d4fbe6b170973 refs/heads/SR\_-1370\_HPCC\_Script\_Performance\_-XpertHRUS

191633c874fcceacdbfe4f2cc82210401578f355 refs/heads/SR\_1516\_HPCC\_to\_Mysql\_ICISNews\_CCIInform\_ICISDB

1946ec1add05751776b6692f6e57da4997f4b3d2 refs/heads/Tableau\_Release\_20171206

433c1551a6afc1e71db255bfa8baba53dcab13ff refs/heads/Updating\_date\_Outputs

bc2ebaf450ef617d27f7ab7eb157776a198c4236 refs/heads/Usage\_Pipeline\_Performance

e372b45a1be34c1900058927728ad61a166e5352 refs/heads/Workbooks\_V0.6

The ls-remote command returns the SHA1 hash of the latest commit for that reference, so it is easy to parse out and get to the exact commit you need if you’re doing some coding. The --heads option lists only branch names since the command can list tags too.

**Add an existing project to GitHub**

1. [Create a new repository](https://help.github.com/articles/creating-a-new-repository) on GitHub. To avoid errors, do not initialize the new repository with README, license, or gitignore files. You can add these files after your project has been pushed to GitHub.
2. Open Git Bash.
3. Change the current working directory to your local project.
4. Initialize the local directory as a Git repository.

git init

1. Add the files in your new local repository. This stages them for the first commit.

git add .

# Adds the files in the local repository and stages them for commit. To unstage a file, use 'git reset HEAD Filename.

1. Commit the files that you've staged in your local repository.

git commit -m "First commit"

# Commits the tracked changes and prepares them to be pushed to a remote repository. To remove this commit and modify the file, use 'git reset --soft HEAD~1' and commit and add the file again.

1. [Push the changes](https://help.github.com/articles/pushing-to-a-remote) in your local repository to GitHub.

git push origin master

# Pushes the changes in your local repository up to the remote repository you specified as the origin

**Additional git commands**

* Use [git log](https://git-scm.com/docs/git-log) to see a log of what has happened so far in the Repository. If you only want to see a specific log, for instance, then add -10 (to show up to 10 commits), --until=2005-12-10 ((to show up to 12/10/2015), etc.
* Use [git checkout](https://git-scm.com/docs/git-checkout) and [git branch](https://git-scm.com/docs/git-branch) to switch branches.
* Use git checkout –b branchname to create a new branch
* Use [git diff](https://git-scm.com/docs/git-diff) and [git status](https://git-scm.com/docs/git-status) to see what you are in the middle of doing.
* Use [git reset](https://git-scm.com/docs/git-reset) and [git checkout](https://git-scm.com/docs/git-checkout) (with pathname parameters) to undo changes.
* Use [git rebase](https://git-scm.com/docs/git-rebase) to maintain topic branches.
* Use [git tag](https://git-scm.com/docs/git-tag) to mark a known point.
* Use git-fetch to download objects and refs from another repository

*git fetch* [<options>] [<repository> [<refspec>…​]]

*git fetch* [<options>] <group>

*git fetch* --multiple [<options>] [(<repository> | <group>)…​]

*git fetch* --all [<options>]

* Use [git help](https://git-scm.com/docs/git-tag) to display help commands as shown below

$ git help

usage: git [--version] [--help] [-C <path>] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

These are common Git commands used in various situations:

start a working area (see also: git help tutorial)

clone Clone a repository into a new directory

init Create an empty Git repository or reinitialize an existing one

work on the current change (see also: git help everyday)

add Add file contents to the index

mv Move or rename a file, a directory, or a symlink

reset Reset current HEAD to the specified state

rm Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)

bisect Use binary search to find the commit that introduced a bug

grep Print lines matching a pattern

log Show commit logs

show Show various types of objects

status Show the working tree status

grow, mark and tweak your common history

branch List, create, or delete branches

checkout Switch branches or restore working tree files

commit Record changes to the repository

diff Show changes between commits, commit and working tree, etc.

merge Join two or more development histories together

rebase Reapply commits on top of another base tip

tag Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)

fetch Download objects and refs from another repository

pull Fetch from and integrate with another repository or a local branch

push Update remote refs along with associated objects

'git help -a' and 'git help -g' list available subcommands and some

concept guides. See 'git help <command>' or 'git help <concept>'

to read about a specific subcommand or concept.

* Use [git help](https://git-scm.com/docs/git-tag) -a to display help commands as shown below

$ git help -a

afekareo@QHSW30702 MINGW64 ~/documents/repohpcc (adb-185\_update\_transaction\_to\_include\_usd\_and\_eur\_values)

$ git help -a

usage: git [--version] [--help] [-C <path>] [-c name=value]

[--exec-path[=<path>]] [--html-path] [--man-path] [--info-path]

[-p | --paginate | --no-pager] [--no-replace-objects] [--bare]

[--git-dir=<path>] [--work-tree=<path>] [--namespace=<name>]

<command> [<args>]

available git commands in 'C:\Program Files\Git\mingw64/libexec/git-core'

add gc receive-pack

add--interactive get-tar-commit-id reflog

am grep remote

annotate gui remote-ext

apply gui--askpass remote-fd

archimport gui--askyesno remote-ftp

archive gui.tcl remote-ftps

askpass hash-object remote-http

bisect help remote-https

bisect--helper http-backend repack

blame http-fetch replace

branch http-push request-pull

bundle imap-send rerere

cat-file index-pack reset

check-attr init rev-list

check-ignore init-db rev-parse

check-mailmap instaweb revert

check-ref-format interpret-trailers rm

checkout log send-email

checkout-index ls-files send-pack

cherry ls-remote sh-i18n--envsubst

cherry-pick ls-tree shortlog

citool mailinfo show

clean mailsplit show-branch

clone merge show-index

column merge-base show-ref

commit merge-file stage

commit-tree merge-index stash

config merge-octopus status

count-objects merge-one-file stripspace

credential merge-ours submodule

credential-manager merge-recursive submodule--helper

credential-store merge-resolve subtree

credential-wincred merge-subtree svn

cvsexportcommit merge-tree symbolic-ref

cvsimport mergetool tag

daemon mktag unpack-file

describe mktree unpack-objects

diff mv update

diff-files name-rev update-git-for-windows

diff-index notes update-index

diff-tree p4 update-ref

difftool pack-objects update-server-info

difftool--helper pack-redundant upload-archive

fast-export pack-refs upload-pack

fast-import patch-id var

fetch prune verify-commit

fetch-pack prune-packed verify-pack

filter-branch pull verify-tag

fmt-merge-msg push web--browse

for-each-ref quiltimport whatchanged

format-patch read-tree worktree

fsck rebase write-tree

fsck-objects rebase--helper