Basic Git Command Line Reference for Windows Users

John Atten

18 Sep 2012 CPOL

Rate this: ★★★★★ 4.88 (21 votes)

CodeProjectWhile there are GUI interfaces available for GIT (some good, some bad), familiarity with at least the basics of git&rsquo;s command line interface can only enhance your ability to use the tool to maximum effectiveness.

CodeProject

git

While there are GUI interfaces available for GIT (some good, some bad), familiarity with at least the basics of git's command line interface can only enhance your ability to use the tool to maximum effectiveness. Since I am relatively new to git and version control in general, I set out to learn the basics of the git command line. In doing so, I found it handy to keep a list of the commonly-used commands nearby so that I didn't have to keep Googling.

In this post, I am going to cover the very basic set of commands one might require to effectively navigate and maintain your source repo using only the git Bash command line interface. Probably, in creating this post, I will not need to look at this again, as the sheer fact of composing this list and explaining it all will burn these into my brain forever. On the other hand, if I am ever unsure, I will now have a place to come look!

NOTE: The references here are by no means comprehensive. I have included the basic commands required to get started, and the commonly used options for each command. There is a wealth of additional information available on the internet, and I have included some helpful reference links at the end of this post.

To more easily find what you might be looking for, here are some links to specific sections of this post:

Working with the file system

Navigating the file system with the Bash Command Line
- Show directory contents with the Bash Command Line
- Create a new directory with the Bash Command Line
- Create files with the Bash Command Line
- Remove files with the Bash Command Line
- Remove directories with the Bash Command Line

Configuring Git and Creating a Repository:

- Configure Git
- Initialize a new repository

Staging and Committing Changes

- Add/Stage files for commit
- Unstage files for commit
- Committing Changes

Working with Remote Repositories (like Github)

- Working with remote repositories
- Working with branches
- Merging Branches
- Pushing changes to remote repositories
- Fetching changes from remote repositories
- Pulling Changes from remote repositories

Undoing Changes and Working With Tags

- Undo changes in the working directory
- Working with Tags

Git Bash: Syntax Notes

Directory Paths

First off, note that Git Bash is a *nix application (Unix/Linux), and expects inputs according to *nix conventions when it comes to file system navigation. This is important when using Git on a windows system, because we need to mentally map the familiar Windows directory notation to Unix format:

Windows Directory Path	**<---- Becomes ----->**	***nix Directory Path**
C:\Users\MyUserFolder\		/c/Users/MyUserFolder/

Strings with Spaces

When we are going to provide an input string with no spaces, we need do nothing. However, strings which contain spaces must be enclosed in quotes. Remember this. Personally, I just use quotes around strings in general.

The “Home” Directory

The file system in *nix systems is set up a little differently than in Windows. Git Bash assumes the existence of a “home” directory for each user. In Windows, the default is your personal user folder. This folder is where Git Bash opens by default. Typing only ***cd*** after the command prompt will always return you to the root level of the home directory.

Command Syntax Format:

The basic command syntax for a git Bash Command is:

Hide Copy Code

```
$ CommandName [options] [directory]
```

In the above, the square brackets denote optional parts of the command. The square brackets themselves are not typed into the command line. Items following a command which are not enclosed in brackets are required.

Cases and Spaces Count

Also note that git Bash is case-sensitive, and spaces count. For Example, the common command to change to another directory is cd. This is NOT the same as CD or Cd.

Optional Input

When portions of a command are optional, we will note this by enclosing them in square braces:

Hide Copy Code

```
$ Command [options]
```

In the above, we do not type the square brackets.

User Input

For our purposes here, when we are presenting command syntax examples, we will denote user-provided values between angle brackets:

Hide Copy Code

```
$ Command [options] <SomeUserInput>
```

In the above, we do not type either the square or angle brackets.

Git Bash: Navigating the File System (cd)

Syntax:

cd [options] [<directory>]

Navigate to the Home Directory (Default folder for the current user):

Hide Copy Code

```
$ cd
```

Navigate to a specific folder in the file system:

Hide Copy Code

```
$ cd /c/SomeFolder/SomeOtherFolder/
```

Navigate to a specific folder in the file system (if there are spaces in the directory path):

Hide Copy Code

```
$ cd “/c/Some Folder/Some Other Folder/”
```

Go back to the previous Location:

Hide Copy Code

```
$ cd -
```

Move Up One Directory Level:

Hide Copy Code

```
$ cd ..
```

In the above, the cd command is followed by a space, then two period with no space between.

Git Bash: Show Directory Contents (ls)

Syntax:

ls [options]

Options:

-1 *= List 1 item per line*

-r *= Reverse the sort order*

-a *= Show Everything, including hidden items*

-d *= list only directories*

-l *= (letter L, lowercase) = Use a long listing format (more info per item, arranged in columns, vertical listing)*

List the contents of the current directory (folder):

Hide Copy Code

```
$ ls
```

The above will display the contents of the current directory as a horizontal list. Not real convenient.

List the contents of the current directory, one item per line:

Hide Copy Code

```
$ ls -1
```

That's better. Note, however, that we can only differentiate files from subdirectories based upon the file extension.

List only the subdirectories (folders) within the current directory:

Hide Copy Code

```
$ ls –d */
```

List everything in long form, vertically:

Hide Copy Code

```
$ ls –al
```

The above gives a swath of information. Also, subdirectories are differentiated by the first column (begin with drwxr instead of -rw)

List all contents, including subdirectory contents, single item per line:

Hide Copy Code

```
$ ls -1 *
```

Git Bash: Create a New Directory (mkdir)

Syntax:

mkdir [options] <folderName>

Options:

-p = *Create parent directories as needed*

--verbose = *Show a message for each new directory created (note the double dash)*

Create a folder in the current directory (without spaces in the folder name):

Hide Copy Code

```
$ mkdir NewFolderName
```

Create a folder in the current directory (with spaces in the folder name):

Hide Copy Code

```
$ mkdir “New Folder Name”
```

Create a folder at the specific directory path:

Hide Copy Code

```
$ mkdir /c/ExistingParentFolder/NewFolderName
```

Create a folder at the specific directory path, and create parent directories as needed:

Hide Copy Code

```
$ mkdir -p /c/NewParentFolder/NewFolderName
```

Create a folder at the specific directory path, create parent directories as needed, and print a description of what was done in the console window:

Hide Copy Code

```
$ mkdir -p --verbose /c/NewParentFolder/NewFolderName
```

Git Bash: Create Files (touch, echo)

Syntax:

touch [options] <FileName>

echo [options] TextString > FileName

(NOTE: FileName can include directory. Default is the current directory).

Create a single (empty) text file in the current directory:

Hide Copy Code

```
$ touch newFile.txt
```

Create a single (empty) text file in the specified directory:

Hide Copy Code

```
$ touch /c/SomeFolder/newFile.txt
```

Create multiple (empty) text files in the current directory:

Hide Copy Code

```
$ touch newFile_1.txt newFile_2 . . . newFile_n
```

Append text to a file. If the file does not exist, one is created:

Hide Copy Code

```
$ echo “This text is added to the end of the file” >> newFile.txt
```

Overwrites text in a file. If the file does not exist, one is created:

Hide Copy Code

```
$ echo “This text replaces existing text in the file” > newFile.txt
```

Overwrites text in a file at the specified location. If the file does not exist, one is created:

Hide Copy Code

```
$ echo “This text replaces existing text in the file” > /c/SomeFolder/newFile.txt
```

Git Bash: Remove Files (rm)

Syntax:

rm [options] -<FileName>

Options:

-I (or --interactive) **=** *Prompt before removal*

-v (or --verbose) **=** *Explain what is being done*

Remove the specified file from the current directory (no spaces):

Hide Copy Code

```
$ rm DeleteFileName
```

Remove the specified file from the current directory (with spaces):

Hide Copy Code

```
$ rm "Delete File Name"
```

Prompt for confirmation before remove the specified file from the current directory (no spaces):

Hide Copy Code

```
$ rm -i DeleteFileName
```

Removes the specified file and reports what was done in the console window:

Hide Copy Code

```
$ rm -v DeleteFileName
```

Git Bash: Remove Directories (rmdir, rm -rf)

Syntax:

rmdir [options] <FolderName>

rm -rf

Removes the specified folder if empty. Operation fails if folder is not empty:

Hide   Copy Code

```
$ rmdir DeleteFolderName
```

Removes the specified folder and all contents:

Hide   Copy Code

```
$ rm -rf DeleteFolderName
```

Git Bash: Configure Git (git config)

Syntax:

git config --global user.name <"User Name">

git config --global user.email <UserEmailAddress>

Set the global User.Name value for the current user on the system:

Hide   Copy Code

```
$ git config --global user.name “FirstName LastName”
```

Set the global User.Email value for the current user on the system:

Hide   Copy Code

```
$ git config --global user.email UserEmailAddress
```

Show me the current values:

The following return the current values for the user.name and user.email properties respectively and output the values to the console window:

Print the current global User.Name value to the console window:

Hide Copy Code

```
$ git config --global user.name
```

Print the current global User.Email value to the console window:

Hide Copy Code

```
$ git config --global user.email
```

Git Bash: Initialize a New Git Repo (git init)

Syntax:

git init

Create files required in the current directory to perform version control:

Hide Copy Code

```
$ git init
```

Git Bash: Add/Stage for Commit (git add)

Syntax:

git add [options] [<File_1>] [<File_2>] . . . [<File_n>]

Options:

-A (or --all) **=** *Add all new or changed files to the staged changes, including removed items (deletions)*

-u **=** *Add changes to currently tracked files and removals to the next commit. Does not add new files.*

. **=** *Adds new or changed files to the staged changes for the next commit, but does not add removals.*

Note that ***git add -A*** is semantically equivalent to ***git add .*** followed by ***git add –u***

-p **=** *Interactive add. Walks through changes in the working directory and prompts for additions*

Add all changes in the working directory to the next commit, including new files and deletions:

Hide Copy Code

```
$ git add -A
```

Add all changes to tracked files and all new files to the next commit, but do not add file deletions:

Hide Copy Code

```
$ git add .
```

adds all changes to tracked files and all file removals to the next commit, but does not add new files:

Hide Copy Code

```
$ git add -u
```

Walks through changed files and prompts user for add option. Does not include new files:

Hide Copy Code

```
$ git add -p
```

Git Bash: Unstage from Commit (git reset)

Syntax:

git reset HEAD <File_1>

Remove the specified file from the next commit:

Hide Copy Code

```
$ git reset HEAD FileName
```

Git Bash: Committing Changes (git commit)

Syntax:

git commit [options] [<File_1>] [<File_2>] . . . [<File_n>] [-m <"Commit Message">]

Options:

-a **=** *Commit all changes to tracked files since last commit*

-v **=** *Verbose: include the diffs of committed items in the commit message screen*

--amend **=** *Edit the commit message associated with the most recent commit*

--amend <File_1> <File_2> . . . <File_n> **=** *redo the previous commit and include changes to specified files.*

Commits the changes for the specific file(s) and includes the commit message specified:

Hide Copy Code

```
$ git commit FileName –m "Message Text"
```

Note that Git requires a commit message. If you do not provide one using the -m option, you will be prompted to do so before the commit is performed.

Commits all files changed since last commit. Does not include new files.

Hide Copy Code

```
$ git commit –a –m "Message Text"
```

Adds all changes to the previous commit and overwrites the commit message with the new Message Text. Does not include new files:

Hide Copy Code

```
$ git commit –a –amend –m "Message Text"
```

Git Bash: Remote Repositories (git remote)

Syntax:

git remote add <RemoteName> <RemoteURL>

git remote show <RemoteName>

NOTE: As used here, RemoteName represents a local ***alias*** *(or nickname) for your remote repository. The name of the remote on the server does not necessarily have to be the same as your local alias.*

Add the specified remote repo to your git config file. The remote can then be pushed to/fetched from:

Hide Copy Code

```
$ git remote add RemoteName https://RemoteName/Proj.git
```

Print information about the specified remote to the console window:

Hide Copy Code

```
$ git remote show RemoteName
```

Git Bash: Branching (git branch)

Syntax:

git branch [options][<BranchName>][<StartPoint>]

Options:

-a = *List all local and remote branches*

-r = *List all remote branches*

List all local branches:

Hide Copy Code

```
$ git branch
```

List all remote branches:

Hide Copy Code

```
$ git branch -r
```

List all local and remote branches:

Hide Copy Code

```
$ git branch -a
```

Create a new branch starting at the some point in history as the current branch:

Hide Copy Code

```
$ git branch BranchName
```

Note that this creates the new branch, but does not "check it out" (make it the current working branch).

Switch from the current branch to the indicated branch:

Hide Copy Code

```
$ git checkout BranchName
```

Create a new branch and switch to it from the current branch:

Hide Copy Code

```
$ git checkout –b NewBranchName StartPoint
```

Note that StartPoint refers to a revision number (or the first 6 characters of such) or an appropriate tag.

Git Bash: Merging Branches

Syntax:

git merge [<BranchName>][--no-commit]

Merge the specified branch into the current branch and auto-commit the results:

Hide Copy Code

```
$ git merge BranchName
```

Merge the specified branch into the current branch and do not commit the results:

Hide Copy Code

```
$ git merge BranchName --no-commit
```

Git Bash: Pushing to Remote Repositories (git push)

Syntax:

git push [<RemoteName> <BranchName>]

Update the remote server with commits for all existing branches common to both the local system and the server. Branches on the local system which have never been pushed to the server are not shared.

Hide Copy Code

```
$ git push
```

Updates the remote server with commits for the specific branch named. This command is required to push a new branch from the local repo to the server if the new branch does not exist on the server.

Hide Copy Code

```
$ git push RemoteName BranchName
```

Git Bash: Fetching from Remote Repositories (git fetch)

Syntax:

git fetch <RemoteName>

Retrieve any commits from the server that do not already exist locally:

Hide Copy Code

```
$ git fetch RemoteName
```

NOTE: git fetch retrieves information from the remote and records it locally as a branch in your current repository. In order to merge the new changes into your local branch, you need to run git fetch followed by git merge. Since there may be more than one branch on the remote repository, it is necessary to specify the branch you wish to merge into your current branch:

Merge syntax for post-fetch merge:

git merge <RemoteName/BranchName>

Merge the newly fetched branch from the remote into your current working branch:

Hide Copy Code

```
$ git merge RemoteName/BranchName
```

Using fetch before merging allows you to pull changesets in from the remote, but examine them and/or resolve conflicts before attempting to merge.

Git Bash: Pulling from Remote Repositories (git pull)

Syntax:

git pull <RemoteName/BranchName>

Fetch changes from the specified branch in the remote, and merge them into the current local branch:

Hide Copy Code

```
$ git pull RemoteName/BranchName
```

NOTE: git pull is essentially the same as running git fetch immediately followed by git merge.

Git Bash: Undo (git reset)

Syntax:

git reset [options]

Options:

--hard **=** *undo everything since the last commit*

--hard ORIG_HEAD = *Undo most recent merge and any changes after.*

--soft HEAD^ = *undo last commit, keep changes staged*

Undo everything since the last commit:

Hide Copy Code

```
$ git reset --hard
```

Undo most recent successful merge and all changes after:

Hide Copy Code

```
$ git reset --hard ORIG_HEAD
```

Undo most recent commit but retain changes in staging area:

Hide Copy Code

```
$ git reset --soft HEAD^
```

Git Bash: Tags (git tag)

Syntax:

git tag [options] [<TagName>] [<CommitChecksum>] [<TagMessage?]

Options:

-a = Annotated Tag

-m = Annotated Tag Message

List all tags in the current repository:

Hide Copy Code

```
$ git tag
```

Create a tag at the current revision:

Hide Copy Code

```
$ git tag TagName
```

Create a tag at the commit specified by the partial checksum (six characters is usually plenty):

Hide Copy Code

```
$ git tag TagName CommitChecksum
```

Create an annotated tag:

Hide Copy Code

```
$ git tag -a TagName -m TagMessage
```

Create an annotated tag at the commit specified by the partial checksum:

Hide Copy Code

```
$ git tag -a TagName CommitChecksum
```

Push tags to a remote repository:

Hide Copy Code

```
$ git push --tags
```

Print information about a specific tag to the console window:

Hide Copy Code

```
$ git show TagName
```

Almost all of the information presented above represents a "toe in the water" sampling designed to get started. There are plenty more commands for use both within Git itself, and from the more general Bash command line. Additionally, most of the commands listed here have more options than I have included. I tried to keep this simple, as a reference for myself, and for whoever else may find it useful to get started with the basics.

Additional Resources

- An A-Z Index of the Bash command line for Linux
- Git Reference
- The Pro Git Book by Scott Chacon (eBook)
- Pro Git (Expert's Voice in Software Development)
- Git Cheat Sheet
- Github
- Help on Github

License

Share

About the Author

John Atten

Software Developer XIV Solutions
United States

My name is John Atten, and my username on many of my online accounts is xivSolutions. I am Fascinated by all things technology and software development. I work mostly with C#, Javascript/Node.js, Various flavors of databases, and anything else I find interesting. I am always looking for new information, and value your feedback (especially where I got something wrong!)

Comments and Discussions

You must Sign In to use this message board.

Spacing Relaxed Layout Normal Per page 25 Update

First Prev Next

The links on the top article	**Member 12342332**	**21-Feb-16 18:55**
Re: The links on the top article	John Atten	21-Feb-16 19:36
Re: The links on the top article	Member 12342332	22-Feb-16 15:23
My vote of 5	**Agent__007**	**7-Oct-14 22:05**
My vote of 5	**Marco Bertschi**	**1-Mar-14 5:27**
Re: My vote of 5	John Atten	1-Mar-14 5:35

Typos in the conventions section	**R. Hoffmann**	**17-Sep-12 23:41**
Re: Typos in the conventions section	John Atten	18-Sep-12 0:58
apart from aforementioned formatting issues ...	**Garth J Lancaster**	**11-Sep-12 12:23**
Re: apart from aforementioned formatting issues ...	John Atten	12-Sep-12 2:40
Formatting and other issues	**Sandip.Nascar**	**11-Sep-12 11:37**
Re: Formatting and other issues	John Atten	12-Sep-12 2:41
Format Issues	**Clifford Nelson**	**11-Sep-12 10:57**
Re: Format Issues	John Atten	12-Sep-12 2:42

Last Visit: 4-Oct-19 0:16 Last Update: 4-Oct-19 0:16 Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.