GIT Command Reference

Official GIT website: https://git-scm.com/

Installing GIT 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. If you're on Fedora/CentOS for example, you can use yum:

```
$ sudo yum install git-all
```

If you're on a Debian-based distribution like Ubuntu, try apt-get:

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

Once installed check GIT version using the following command:

```
$ git --version
  Or
$ git -v
```

1. Initial Configuration

Set Your Identity

The first thing you should do when you install Git is to set your user name and email address. This is important because every Git commit uses this information, and it's immutably baked into the commits you start creating:

```
$ git config --global user.name "Firstname Lastname"
$ git config --global user.email myname@mycompany.com
```

Again, you need to do this only once if you pass the `--global` option, because then Git will always use that information for anything you do on that system. If you want to override this with a different name or email address for specific projects, you can run the command without the `--global` option when you're in that project.
Many of the GUI tools will help you do this when you first run them.

Set your default Editor

Now that your identity is set up, you can configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your system's default editor.

If you want to use a different text editor, such as Emacs, you can do the following:

```
$ git config --global core.editor emacs
```

While on a Windows system, if you want to use a different text editor, such as Notepad++, you can do the following:

On a x86 system

```
$ git config --global core.editor "'C:/Program
Files/Notepad++/notepad++.exe' -multiInst -nosession"
```

On a x64 system

```
$ git config --global core.editor "'C:/Program Files (x86)/Notepad++/
notepad++.exe' -multiInst -nosession"
```

Checking Your Settings

Use `git config --list` command to list all the settings Git can find at that point:

```
$ git config --list
user.name=John Doe
user.email=johndoe@example.com
color.status=auto
color.branch=auto
color.interactive=auto
color.diff=auto...
```

You may see keys more than once, because Git reads the same key from different files (`/etc/gitconfig` and `~/.gitconfig`, for example). In this case, Git uses the last value for each unique key it sees.
You can also check what Git thinks a specific key's value is by typing `git config <key>`:

```
$ git config user.name
John Doe
```

2. Creating a Git Repository

You can get a Git project using two main approaches. The first takes an existing project or directory and imports it into Git. The second clones an existing Git repository from another server.

Initializing a Repository in an Existing Directory

If you're starting to track an existing project in Git, you need to go to the project's directory and type:

```
$ git init
```

Cloning and Existing Repository

```
$ git clone https://github.com/libgit2/libgit2 (http protocol)
$ git clone user@server:path/to/repo.git (ssh protocol)
```

That creates a directory named "libgit2", initializes a `.git` directory inside it, pulls down all the data for that repository, and checks out a working copy of the latest version. If you go into the new `libgit2`directory, you'll see the project files in there, ready to be worked on or used.

3. GIT Workflow (Local)

Once you have cloned or initialized a new GIT project, begin changing files as needed. There is no locking of files or a traditional VCS checkout concept. Simply begin editing files in a progression towards a committable state.

Adding (staging)

New files must be 'staged' with the add command as follows:

```
$ git add index.html
$ git add javascript/
$ git add *.js
```

Committing

Once all desired files are added and staged, a commit command saves the pending additions to the local repository. The default text $EDITOR will be opened for entry of the commit message.

```
$ git commit
$ git commit -m "Your commit message"
```

Status

To check the current status of a project's local directories and files, such as modified, new, deleted or Untracked files, invoke the following command:

```
$ git status
```

Output:

```
$ git status
 On branch master
 Initial commit
 Changes to be committed:
   (use "git rm --cached <file>..." to unstage)
         new file:   text1.txt
         new file:   text2.txt
```

Branching

Concept of branching in GIT is very similar to any other SCM tool. Following are some of the branching commands:

```
$ git branch –a (shows all branches along with current one)
$ git branch <new branch name> (create a new branch)
$ git checkout <branch_name> (checkout a branch)
$ git checkout –b <new branch name> create a new branch and checkout
that branch in a single command
```

Merging

Like other popular version control systems, git allows you to merge one or more branches into the current branch.

```
$ git merge <branch one>
$ git merge <branch one> <branch two>
```

4. GIT Workflow (Remote Collaboration)

Working with remote repositories is one of the primary features of GIT. You can push or pull, depending on your desired workflow with colleagues and based on the repository operating system file and protocol permissions.

Remotes

While full URLs to other repositories can be specified as a source or destination for the majority of GIT commands, this quickly becomes unwieldy and a shorthand solution is called for. These bookmarks of other repository locations are called ***remotes***.

Full addresses of your configured ***remotes*** can be viewed with:

```
$ git remote -v
```

To add a new remote, type:

```
$ git remote add <remote name> <remote address>
```

Push

Pushing with GIT is the transmission of local changes to a colleague or community repository with sufficiently open permissions to allow you to write to it.

```
$ git push <remote name> <branch name>
$ git push <remote name> <local branch name:remote branch name>
```

The push command performs a publishing action, and sends the GIT commit history of one or more branches to an upstream GIT repository. Pushed branches are then accessible to anyone with access to this remote GIT repository.

Pull

The action of a GIT pull consists of the combination of retrieving (fetching) a remote repository's contents and automatically merging the file changes and commit history into the current branch.

```
$ git pull
$ git pull origin
```

Fetch

An alternative to pulling content, which automatically merges inbound changes with your local history, is to retrieve the remote (upstream) changes and store the content conveniently in a cache for evaluation or selective merging.

```
$ git fetch <remotename>
$ git merge <remotename/branchname>
```

5. Viewing the Commit History

After you have created several commits, or if you have cloned a repository with an existing commit history, you'll probably want to look back to see what has happened. The most basic and powerful tool to do this is the '***git log'*** command.

When you run git log in a project, you should get output that looks something like this:

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
    changed the version number
commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700
    removed unnecessary test
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700
    first commit
```

By default, with no arguments, git log lists the commits made in that repository in reverse chronological order – that is, the most recent commits show up first. As you can see, this command lists each commit with its SHA-1 checksum, the author's name and email, the date written, and the commit message.

A huge number and variety of options to the git log command are available to show you exactly what you're looking for. Here, we'll show you some of the most popular.

One of the more helpful options is -p, which shows the difference introduced in each commit. You can also use -2, which limits the output to only the last two entries:

```
$ git log -p -2
```

You can also use a series of summarizing options with git log. For example, if you want to see some abbreviated stats for each commit, you can use the --stat option:

```
$ git log --stat
```

Table 1: Common options to `git log`

Option	Description
`-p`	Show the patch introduced with each commit.
`--stat`	Show statistics for files modified in each commit.
`--shortstat`	Display only the changed/insertions/deletions line from the --stat command.
`--name-only`	Show the list of files modified after the commit information.
`--name-status`	Show the list of files affected with added/modified/deleted information as well.
`--abbrev-commit`	Show only the first few characters of the SHA-1 checksum instead of all 40.
`--relative-date`	Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.
`--graph`	Display an ASCII graph of the branch and merge history beside the log output.
`--pretty`	Show commits in an alternate format. Options include oneline, short, full, fuller, and format (where you specify your own format).

I strongly urge you to try out all the options to understand each sub command for git log and what it is capable of.

6. Limiting Log Output

In addition to output-formatting options, git log takes a number of useful limiting options – that is, options that let you show only a subset of commits. You've seen one such option already – the -2 option, which show only the last two commits. In fact, you can do -<n>, where n is any integer to show the last n commits. In reality, you're unlikely to use that often, because Git by default pipes all output through a pager so you see only one page of log output at a time.

However, the time-limiting options such as |--since| and |--until| are very useful. For example, this command gets the list of commits made in the last two weeks:

```
$ git log --since=2.weeks
```

This command works with lots of formats – you can specify a specific date like |"2008-01-15"|, or a relative date such as |"2 years 1 day 3 minutes ago".

Table 2: Options to limit the output of `git log`

Option	Description
`-(n)`	Show only the last n commits
`--since`, `--after`	Limit the commits to those made after the specified date.
`--until`, `--before`	Limit the commits to those made before the specified date.
`--author`	Only show commits in which the author entry matches the specified string.
`--committer`	Only show commits in which the committer entry matches the specified string.
`--grep`	Only show commits with a commit message containing the string
`-S`	Only show commits adding or removing code matching the string