GitPython Quick Start Tutorial

Welcome to the GitPython Quickstart Guide! Designed for developers seeking a practical and interactive learning experience, this concise resource offers step-by-step code snippets to swiftly initialize/clone repositories, perform essential Git operations, and explore GitPython's capabilities. Get ready to dive in, experiment, and unleash the power of GitPython in your projects!

All code presented here originated from ***** insert link **** to assure correctness. Knowing this should also allow you to more easily run the code for your own testing purposes. All you need is a developer installation of git-python.

git.Repo

There are a few ways to create a `git.Repo` object

An existing local path

$ git init path/to/dir

```
from git import Repo

repo = Repo.init(path_to_dir)  # git init path/to/dir
```

Existing local git Repo

```
repo = Repo(path_to_dir)
```

Clone from URL

For the rest of this tutorial we will use a clone from https://github.com/gitpython-developers/QuickStartTutorialFiles.git

$ git clone https://github.com/gitpython-developers/QuickStartTutorialFiles.git

```
repo_url = "https://github.com/gitpython-developers/QuickStartTutorialFiles.git"

repo = Repo.clone_from(repo_url, local_dir)
```

Usage

- $ git add filepath

```
# We must make a change to a file so that we can add the update to git

update_file = 'dir1/file2.txt' # we'll use local_dir/dir1/file2.txt
with open(f"{local_dir}/{update_file}", 'a') as f:
    f.write('\nUpdate version 2')
```

Now lets add the updated file to git

```
add_file = [update_file]  # relative path from git root
repo.index.add(add_file)  # notice the add function requires a list of paths
```

Notice the add method requires a list as a parameter

- $ git commit -m message

```
repo.index.commit("Update to file2")
```

- $ git log file

A list of commits associated with a file

```
# relative path from git root
repo.iter_commits(all=True, max_count=10, paths=update_file)  # gets the last 10 commits
from all branches

# Outputs: <generator object Commit._iter_from_process_or_stream at 0x7fb66c186cf0>
```

Notice this returns a generator object

```
commits_for_file_generator = repo.iter_commits(all=True, max_count=10, paths=update_file)
commits_for_file = [c for c in commits_for_file_generator]
commits_for_file

# Outputs: [<git.Commit "SHA1-HEX-HASH-1">,
# <git.Commit "SHA1-HEX-HASH-1">]
```

returns list of `Commit` objects

- $ git status

- Untracked files

Lets create a new file

```
# We'll create a file5.txt

with open(f'{local_dir}/file5.txt', 'w') as f:
    f.write('file5 version 1')
```

```
repo.untracked_files
# Output: ['file5.txt']
```

- Modified files

```
# Lets modify one of our tracked files

with open(f'{local_dir}/Downloads/file3.txt', 'w') as f:
    f.write('file3 version 2')  # overwrite file 3
```

```
repo.index.diff(None)  # compares staging area to working directory
repo.index.diff(repo.head.commit)  # compares staging area to last commit
# Output: [<git.diff.Diff object at 0x7fb66c076e50>,
# <git.diff.Diff object at 0x7fb66c076ca0>]
```

returns a list of `Diff` objects

```
diffs = repo.index.diff(None)
for d in diffs:
    print(d.a_path)

# Downloads/file3.txt
# file4.txt
```

Trees & Blobs

Latest Commit Tree

```
tree = repo.head.commit.tree
```

Any Commit Tree

```
prev_commits = [c for c in repo.iter_commits(all=True, max_count=10)]  # last 10 commits
from all branches
tree = prev_commits[0].tree
```

Display level 1 Contents

```
files_and_dirs = [entry for entry in tree]
files_and_dirs

# Output
# [<git.Commit "SHA1-HEX-HASH-1">,
# <git.Commit "SHA1-HEX-HASH-1">,
# <git.Commit "SHA1-HEX-HASH-1">]
```

Recurse through the Tree

```
def print_files_from_git(root, delim='-', i=0):
    for entry in root:
        print(f'{delim if i != 0 else ""}| {entry.path}, {entry.type}')
        if entry.type == "tree":
            print_files_from_git(entry, delim * 4, i + 1)
```

```
print_files_from_git(tree)

# Output
# | Downloads, tree
# ---- | Downloads / file3.txt, blob
# | dir1, tree
# ---- | dir1 / file1.txt, blob
# ---- | dir1 / file2.txt, blob
# | file4.txt, blob
```

Printing text files

```
print_file = 'dir1/file2.txt'
tree[print_file]

# Output <git.Blob "SHA1-HEX-HASH-1">
```

```
blob = tree[print_file]
print(blob.data_stream.read().decode())

# Output
# file 2 version 1
# Update version 2
```