1 Introduction

Git is a version control software, meaning it can manage changes to a project without overwritting its existing data. If you've been coding for a while, you often find yourself saving essentially almost the same code onto another file when making changes/updates/debugs. Well, not anymore! With Github, you are able to create branches that can later be merged with your master code, in addition, Git keeps a snapshot of every change made for refenrence. GitHub is an online service that provides Git respitory hosting. GitHub's main attraction is its collaborative tool components, making team projects productive and efficient. This feature is extremely useful for team projects, since anyone in the team can clone the code and work on different parts simultaneously. Anyone is the team can run tests, fix bugs, contribute new code all while knowing every version can be recovered at any time. GitHub it is also the corner stone of open source code, there is so much you can learn.

2 Terminology

Respitory: This encompases your entire collection of files (text, code, images, you name it..) that is associated with a respective project. This can be stored either locally or in an online host such as GitHub.

Commit: This is a command that takes a snapshot of your respitory at that specific time. This serves as a checkpoint or backup that you may return to if you decide you took a wrong path in your coding progress. The commit comes with a detailed description of the changes made and why.

Branch: When collaborating, in order to work on a project simultaneously, a member may branch off the main code – called the Master Branch – which clones the code, enabling anyone to make updates or edits that may later be merged onto the master branch if approved by the respitory administrator.

Clone: You may also obtain a copy of the respitory on your local computer, rather than an online host. This allows you to make changes on your preferred text editor. When you are satisfied with the changes, you can push the commit to a remote server.

Push: This is a command that allows you to push, or send the changes you have made in your local computer to the remote respitory.

Issue: This refers to a suggested update/edit, an assignment or a question relating to the respitory. Anyone is GitHub is able to create an issue. Each issue features a discussion forum which may be assigned to respitory collaborators.

Pull Request: A request from an user that proposes changes to a respitory, which may be denied or granted by respitory collaborators.

Merge: When a pull request is accepted, the changes from a branch is merged onto another (doesn't have to be the master branch) which may be done via GitHub or from the command line.

3 Getting Started

3.1 Creating a Local Git Respitory

We will be using Terminal to create a local respitory (also known as repo). Let's say we want to name our respitory Cyclotron. We use `mkdir` command to create a new directory (folder). The `~/` is there to ensure that we build this directory at the top level of your computer's file structure. Although you may build this on your desktop directory, it is not recommended since there is only one allowed repo per directory.

```
1 mkdir ~/Cyclotron
```

```
cd ~/Cyclotron
```

and then you procede to initialize the git respitory into the root of your new folder.

```
git init
```

You will see something like Initialized empty Git repository in /Users/Tony/Cyclotron/.git/ with the appropriate user's name, of course.

3.2 Adding/Creating Files to Local Repo

To add an existing file into your local repo, you may simply move the file with the mv command; say you have an existing file name cyclotron.py on your desktop. To move your files, you have to first specify the path of your file followed by a space and the path you want to move your file into.

```
mv ~/Desktop/cyclotron.py ~/Cyclotron/
```

You can also create a new file by running the touch command followed by the desired name and extension.

```
touch RK4.tex
```

Now, if you list your files with the ls command, you can clearly see your files within your folder. However, at this point **git will not recognize your file**. To see what I mean, type the following:

```
git status
```

This command displays any modified or created file in the working directory. The output will look something like this:

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  cyclotron.py

nothing added to commit but untracked files present (use "git add" to track)
```

Notice that we have an untracked file named cyclotron.py, meaning Git is ignoring it for the moment. You can instruct Git to recognize your file with the git add command. Afterwards, use the status command to notice the changes.

```
git add cyclotron.py
git status
```

In order to take a snapshot of your current project, or "commit", you use git commit command along with a description and its purpuse inside quotation marks.

```
git commit -m "Description of the file"
```

3.3 Repo Branches

Say you want to make an update and add a new feature, but you don't want to risk disturbing your original data. You can then make a branch that may be later be merged with the master branch once you are satisfied with your edits. Let's pretend you want to test the Euler Method rather than the RK4, then you can run the following code:

```
 git checkout -b Euler_Testing
```

You may confirm you created a new branch with the git branch command. The asterisk (*) indicates which branch you are currently in.

3.4 Online Respitory on GitHub

Since we are interested in working in teams, we want to add our respitory to a remote server like GitHub. From your GitHub home page, click the New Respitory button, fill out the prompted information and click Create Respitory. I should point out that is it highly recommended to keep the name of you local and remote repo the same.

We are now in the position to push our local respitory onto GitHub. Type and execute the following:

```
git remote add origin https://github.com/TonyAlarcon/Cyclotron.git
git push -u origin master
```

Note that you must change the URL in the first line to the URL displayed by GitHub, which include your username and repo name.

Below is a command cheat sheet obtained online.

GitHub
GIT CHEAT SHEET

Git is the free and open source distributed version control system that's responsible for everything GitHub related that happens locally on your computer. This cheat sheet features the most important and commonly used Git commands for easy reference.

INSTALLATION & GUIS

With platform specific installers for Git, GitHub also provides the ease of staying up-to-date with the latest releases of the command line tool while providing a graphical user interface for day-to-day interaction, review, and repository synchronization.

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

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

For Linux and Solaris platforms, the latest release is available on the official Git web site.

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

SETUP

Configuring user information used across all local repositories

`git config --global user.name "[firstname lastname]"`
set a name that is identifiable for credit when review version history
`git config --global user.email "[valid-email]"`
set an email address that will be associated with each history marker
`git config --global color.ui auto`
set automatic command line coloring for Git for easy reviewing

SETUP & INIT

Configuring user information, initializing and cloning repositories

`git init`
initialize an existing directory as a Git repository
`git clone [url]`
retrieve an entire repository from a hosted location via URL

STAGE & SNAPSHOT

Working with snapshots and the Git staging area

`git status`
show modified files in working directory, staged for your next commit
`git add [file]`
add a file as it looks now to your next commit (stage)
`git reset [file]`
unstage a file while retaining the changes in working directory
`git diff`
diff of what is changed but not staged
`git diff --staged`
diff of what is staged but not yet committed
`git commit -m "[descriptive message]"`
commit your staged content as a new commit snapshot

BRANCH & MERGE

Isolating work in branches, changing context, and integrating changes

`git branch`
list your branches. a * will appear next to the currently active branch
`git branch [branch-name]`
create a new branch at the current commit
`git checkout`
switch to another branch and check it out into your working directory
`git merge [branch]`
merge the specified branch's history into the current one
`git log`
show all commits in the current branch's history

INSPECT & COMPARE

Examining logs, diffs and object information

`git log`	show the commit history for the currently active branch
`git log branchB..branchA`	show the commits on branchA that are not on branchB
`git log --follow [file]`	show the commits that changed file, even across renames
`git diff branchB...branchA`	show the diff of what is in branchA that is not in branchB
`git show [SHA]`	show any object in Git in human-readable format

SHARE & UPDATE

Retrieving updates from another repository and updating local repos

`git remote add [alias] [url]`	add a git URL as an alias
`git fetch [alias]`	fetch down all the branches from that Git remote
`git merge [alias]/[branch]`	merge a remote branch into your current branch to bring it up to date
`git push [alias] [branch]`	Transmit local branch commits to the remote repository branch
`git pull`	fetch and merge any commits from the tracking remote branch

TRACKING PATH CHANGES

Versioning file removes and path changes

`git rm [file]`	delete the file from project and stage the removal for commit
`git mv [existing-path] [new-path]`	change an existing file path and stage the move
`git log --stat -M`	show all commit logs with indication of any paths that moved

REWRITE HISTORY

Rewriting branches, updating commits and clearing history

`git rebase [branch]`	apply any commits of current branch ahead of specified one
`git reset --hard [commit]`	clear staging area, rewrite working tree from specified commit

IGNORING PATTERNS

Preventing unintentional staging or commiting of files

```
logs/
*.notes
pattern*/
```

Save a file with desired patterns as .gitignore with either direct string matches or wildcard globs.

`git config --global core.excludesfile [file]`	system wide ignore pattern for all local repositories

TEMPORARY COMMITS

Temporarily store modified, tracked files in order to change branches

`git stash`	Save modified and staged changes
`git stash list`	list stack-order of stashed file changes
`git stash pop`	write working from top of stash stack
`git stash drop`	discard the changes from top of stash stack