Key URLs

- [http://www.github.com/GreenGroup](http://www.github.com/GreenGroup)
  - Git repository of all RMG-Py code
- [http://greengroup.github.io/RMG-Py/](http://greengroup.github.io/RMG-Py/)
  - Online version of the current RMG-Py documentation
- [http://rmg.mit.edu](http://rmg.mit.edu)
  - Official RMG-Py documentation, thermodynamics and kinetics database browser, and web tools
- [http://dev.rmg.mit.edu](http://dev.rmg.mit.edu)
  - Developmental version of [rmg.mit.edu](http://rmg.mit.edu) with latest features and potential bugs
  - To use, add 18.172.0.124 [dev.rmg.mit.edu](http://dev.rmg.mit.edu) to hosts file in your operating system
Git

- Git is a version control tool
  - Multiple users can edit multiple copies of code
  - Single user can create multiple branches for a single repository
- Online detailed tutorial:
- Where to find programs to help you use git:
Getting started: create a local repo

Two common scenarios: (only do one of these)

a) To **clone an already existing repo** to your current directory:

$ git clone <url> [local dir name]

This will create a directory named *local dir name*, containing a working copy of the files from the repo, and a `.git` directory (used to hold the staging area and your actual repo)

b) To **create a Git repo** in your current directory:

$ git init

This will create a `.git` directory in your current directory. Then you can commit files in that directory into the repo:

$ git add file1.java

$ git commit -m “initial project version”
Basic Git Workflow

1. **Modify** files in your working directory.
2. **Stage** files, adding snapshots of them to your staging area.
3. Make a **commit**, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.
Git file lifecycle

Notes:
If a particular version of a file is in the git directory, it’s considered committed.
If it’s modified but has been added to the staging area, it is staged.
If it was changed since it was checked out but has not been staged, it is modified.
Local Commits

1. ‘git status’ to check which files are modified
   -’git diff <filename>’ shows line-by-line changes
2. ‘git add <filename>’ stages all desired files
3. ‘git commit’ creates new snapshot of staged files and adds to the history
4. ‘git log’ pulls up history of branch; should see your latest commit
Writing Commit Messages

- **First line** is <80 character summary
- Followed by **detailed description**
  - List of all additions/changes
  - Motivation
  - Implementation details
- Examples of bad git messages:
  - “Typo”
  - “Add database entries”
- After saving message a unique commit string is created for each entry
Git Branches

- Branches start a new history to make experimental features

- Allows experimentation without fear of “messing up the code”
Commands used with Branches

- Git branch: pulls up a list of all the branches
- Git branch <new branch>: forks a new from the current head

From this commit:
Git branch nice_feature
Commands used with Branches

- **Git checkout `<location>`**: moves the head to location (can be a commit string or branch name)
- **Git merge `<branch>`**: merges all commits from branch

Git checkout abcd1234 brings you here from anywhere else

From here:
- Git merge nice_feature
  - Commits the nice feature to master
Advanced History Control: Rebase

- Normally when merging: make a new commit that incorporates all changes

- From experiment: Git rebase master merges and chronologically reorders commits
Full Control: Git rebase interactive

- Git rebase –i <commit string>: opens interactive GUI that allows full rewriting of history
  - Delete or reorder commits
  - Squash commits together
  - Make changes to a commit
  - Rewrite commit messages

**WARNING**: do not use this to rewrite history you have pushed to official
Green Group Repos

Official Green Group Repo
- [https://github.com/GreenGroup/](https://github.com/GreenGroup/)
- For official distribution

Personal Github Repo
- Sharing developments with other users
- Back up of in-progress code

Local Repositories
- Running jobs
- Code development and debugging

Git pull

Git push

Pull Request
Setting up Remote Repos

- Git remote add <remote name> <url>

- If you originally forked from GreenGroup official:
  - Git remote rename origin official
  - Create your own fork on Github and name origin
Pulling/Pushing Commits

- Each repo has its own branches
- Commands for pulling and pushing call branches
  - Most common call: `git pull official master`
  - For pushing: `git push origin new_feature`
- Good idea to try to keep branch names consistent
Keeping Official Repo Clean

- To push a commit to official:
  1. Clean up your commit history with `Git rebase -i <first new commit>`
  2. Check that your current commit is updated up to the official `Git pull --rebase official master`
  3. Push to your personal GitHub repo: `Git push origin new_feature`
  4. Make formal **pull request** from your GitHub Repo
# Common Git commands

<table>
<thead>
<tr>
<th>command</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>git clone <code>url [dir]</code></td>
<td>copy a git repository so you can add to it</td>
</tr>
<tr>
<td>git add <code>files</code></td>
<td>adds file contents to the staging area</td>
</tr>
<tr>
<td>git commit</td>
<td>records a snapshot of the staging area</td>
</tr>
<tr>
<td>git status</td>
<td>view the status of your files in the working directory and staging area</td>
</tr>
<tr>
<td>git diff</td>
<td>shows diff of what is staged and what is modified but unstaged</td>
</tr>
<tr>
<td>git help <code>[command]</code></td>
<td>get help info about a particular command</td>
</tr>
<tr>
<td>git pull</td>
<td>fetch from a remote repo and try to merge into the current branch</td>
</tr>
<tr>
<td>git push</td>
<td>push your new branches and data to a remote repository</td>
</tr>
</tbody>
</table>

Others: init, reset, branch, checkout, merge, log, tag

---

18
You can do all of this using Git-cola: a powerful GUI interface

Commit or revert specific lines. Stage files and write commit messages graphically. Amend commits.
Git-cola: a powerful GUI interface

Available for Windows, Mac, and Linux!

Visualize past commit history and repository branches. (Great for tracking specific code changes.)