# A primer on modern version control

Timothy Smith

ECCO Summer School
May 27, 2019

# What is git?

A program accessed through the command line or through a Graphical User Interface (GUI) which provides:

# What is git?

A program accessed through the command line or through a Graphical User Interface (GUI) which provides:

- a way to manage updates to your code, writing, scripts, ...

```
myfile.F

myfile_fixed.F

myfile_fixed_faster.F

myfile_fixed_faster_reallyThisTime.
```

# What is git?

A program accessed through the command line or through a Graphical User Interface (GUI) which provides:

- a way to manage updates to your code, writing, scripts, ...
- a method for contributing to existing projects, mainly software (for LaTeX, see overleaf)

```
myfile.F
myfile_fixed.F
myfile_fixed_faster.F
myfile_fixed_faster_reallyThisTime.
```

# What is git?

A program accessed through the command line or through a Graphical User Interface (GUI) which provides:

- a way to manage updates to your code, writing, scripts, ...
- a method for contributing to existing projects, mainly software (for LaTeX, see overleaf)
- a method for managing file(s) across multiple machines (vs secure copying these files across servers)

```
myfile.F

myfile_fixed.F

myfile_fixed_faster.F

myfile_fixed_faster_reallyThisTime.
```

# Where is my stuff?

- Companies like GitHub and BitBucket "host" your files in the cloud, and track changes as commits (discussed later)

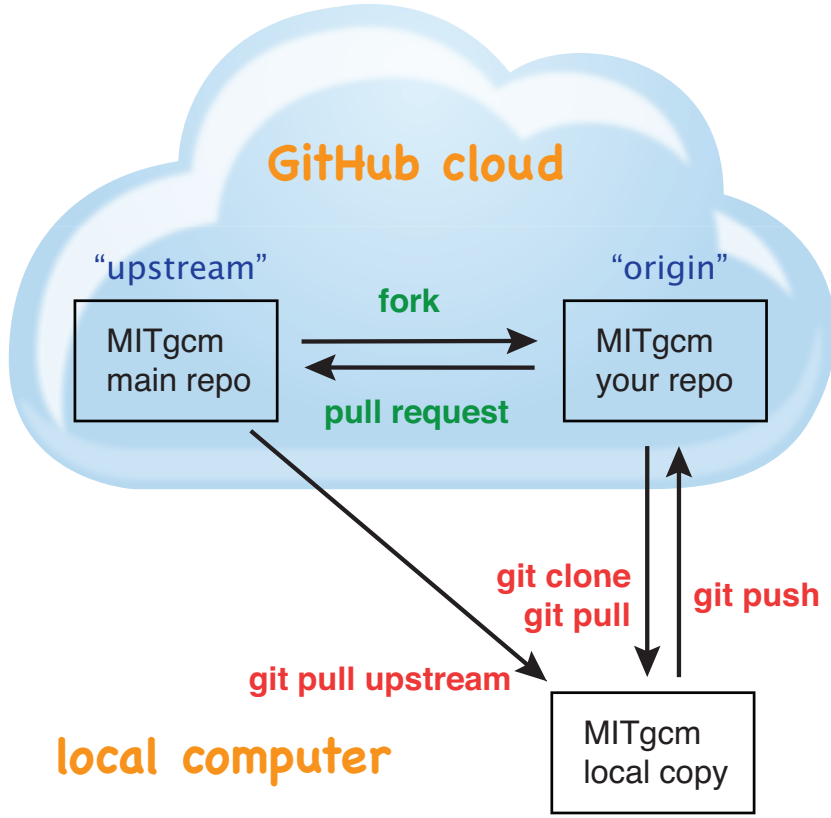



Figure: from the MITgcm documentation

# Where is my stuff?

- Companies like GitHub and BitBucket "host" your files in the cloud, and track changes as commits (discussed later)
- Users make a repository (repo) which is a collection of these files

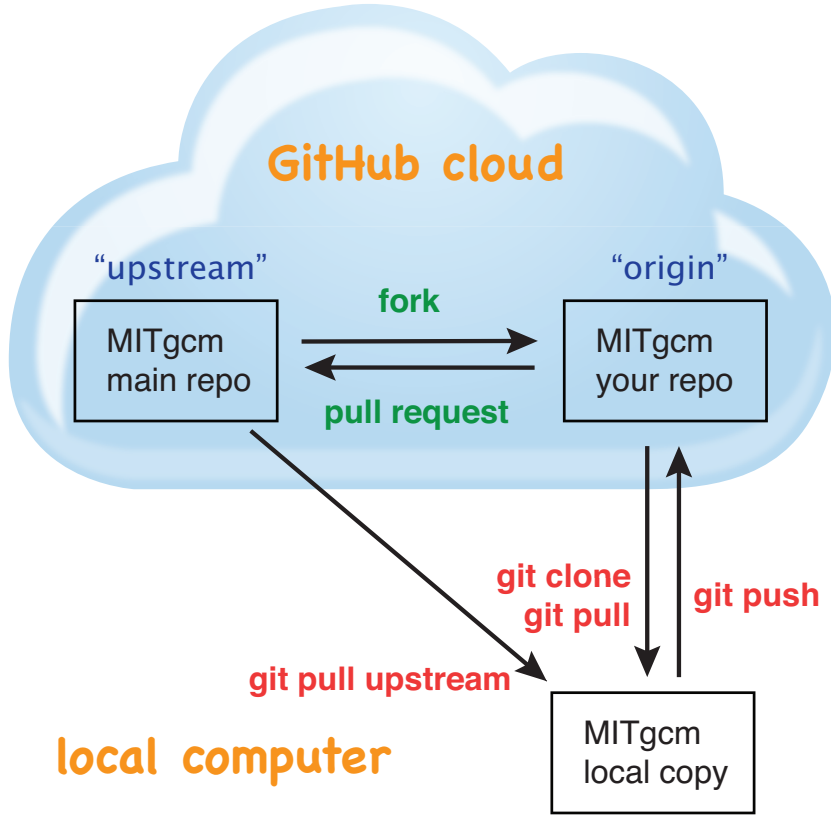



Figure: from the MITgcm documentation

# Where is my stuff?

- Companies like GitHub and BitBucket "host" your files in the cloud, and track changes as commits (discussed later)
- Users make a repository (repo) which is a collection of these files
- Either through command line or GUI, "publish" changes to these docs to the cloud

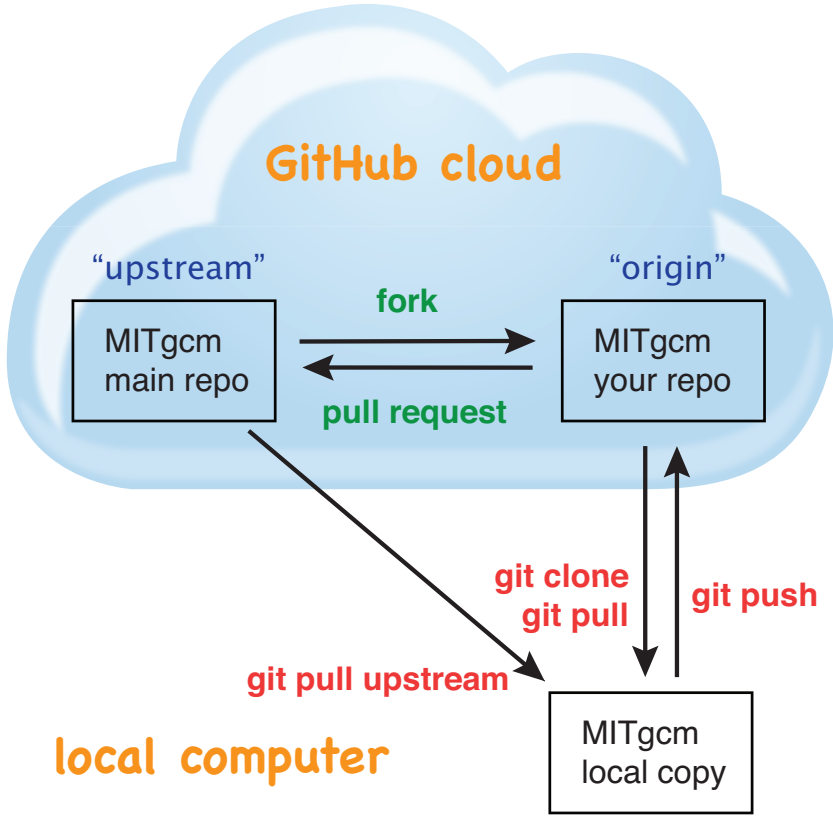



Figure: from the MITgcm documentation

# Where is my stuff?

- Companies like GitHub and BitBucket "host" your files in the cloud, and track changes as commits (discussed later)
- Users make a repository (repo) which is a collection of these files
- Either through command line or GUI, "publish" changes to these docs to the cloud
- Hard to track changes to pdf, figures, ipynb, binaries

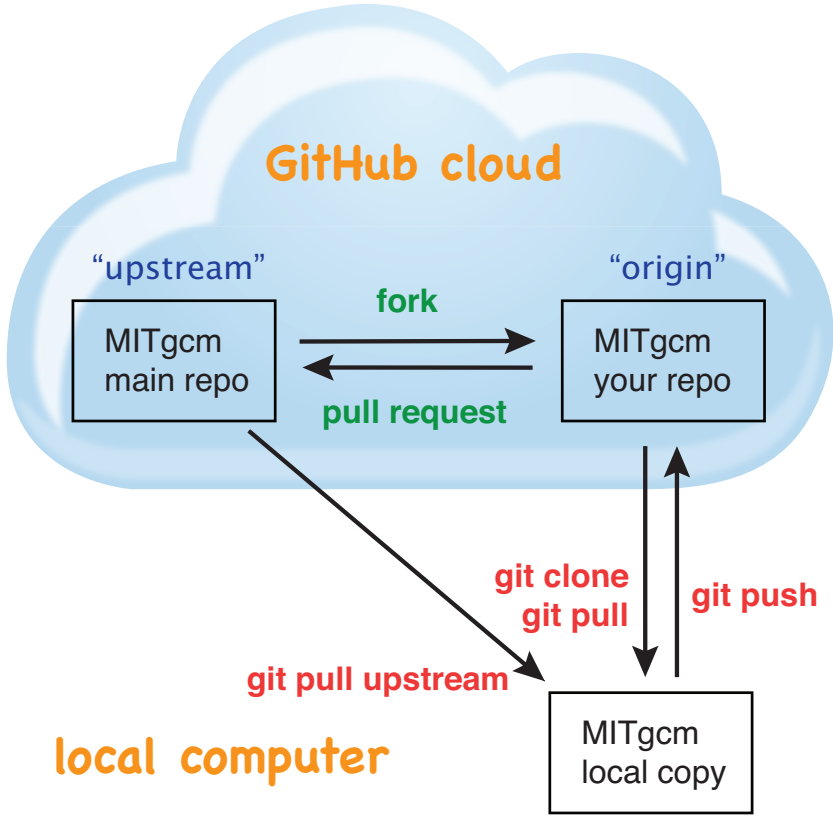



Figure: from the MITgcm documentation

# Where is my stuff?

- Companies like GitHub and BitBucket "host" your files in the cloud, and track changes as commits (discussed later)
- Users make a repository (repo) which is a collection of these files
- Either through command line or GUI, "publish" changes to these docs to the cloud
- Hard to track changes to pdf, figures, ipynb, binaries
- See also mercurial or (deprecated) svn, cvs

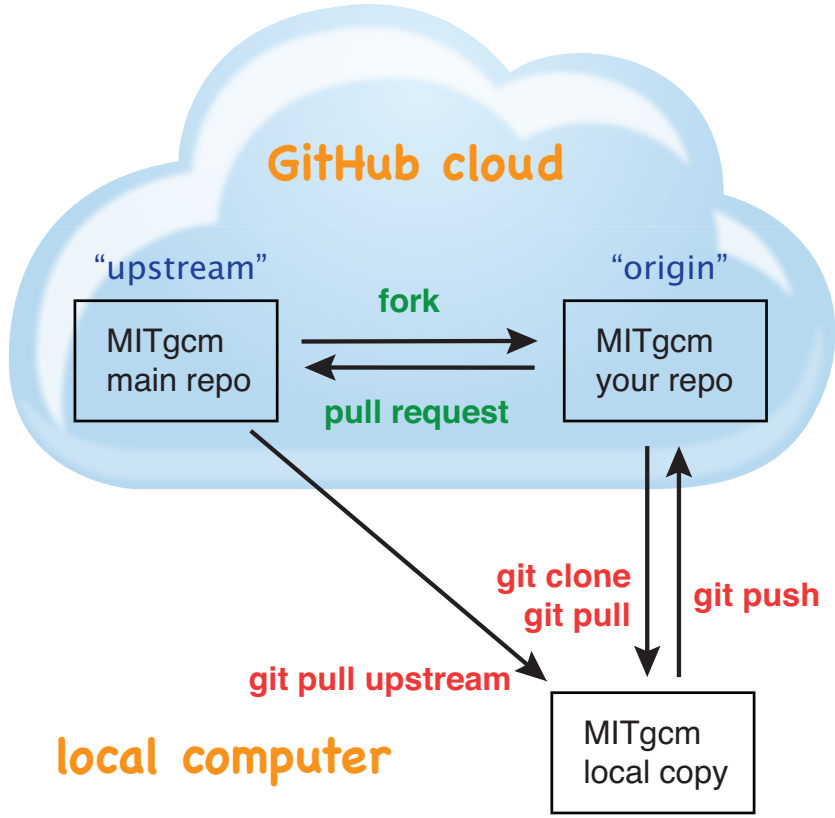



Figure: from the MITgcm documentation

# What to use `git` for?

- scratch repo for pre- or post-processing scripts, like python or MATLAB

# What to use `git` for?

- scratch repo for pre- or post-processing scripts, like python or MATLAB
- new projects, especially collaborative (free account: only 3 collaborators)

# What to use `git` for?

- scratch repo for pre- or post-processing scripts, like python or MATLAB
- new projects, especially collaborative (free account: only 3 collaborators)
- bash environment scripts (e.g. `.bashrc`) across multiple machines, e.g. bash-envy

# What to use `git` for?

- scratch repo for pre- or post-processing scripts, like python or MATLAB
- new projects, especially collaborative (free account: only 3 collaborators)
- bash environment scripts (e.g. `.bashrc`) across multiple machines, e.g. bash-envy
- your personal website, e.g. `https://julianmak.github.io/index.html`

# What to use `git` for?

- scratch repo for pre- or post-processing scripts, like python or MATLAB
- new projects, especially collaborative (free account: only 3 collaborators)
- bash environment scripts (e.g. `.bashrc`) across multiple machines, e.g. bash-envy
- your personal website, e.g. `https://julianmak.github.io/index.html`
- personal written manuscripts or presentations, such as your thesis or this presentation, particularly for LaTex users

# git greatest hits: log

`git log`

```
commit 3512573755525bb8f9dab9a7ac2318d70e2505f80 (HEAD -> master, u
pstream/master, origin/master, origin/HEAD)
Author: Timothy Smith <timsmith204@utexas.edu>
Date:   Mon May 27 11:26:01 2019 -0700

    Added stampede2 optfiles (#246)

    * Added stampede2 optfiles
    * remove trailing blank
    * rename mpi->impi
    * Remove "-DALWAYS_USE_MPI"
      ALWAYS_USE_MPI has been (completely) removed from the code.

commit 914da317ef24c69f5448c2f0bf4043271c2cc982
Author: jm-c <jm-c@users.noreply.github.com>
Date:   Fri May 24 23:53:59 2019 -0400

    Quick fix to PR #242 (#247)

    * fix so that it compiles with SEAICE_ITD defined
    * fix for experiment offline_cheapaml to compile
      Need to have S/R arguments to be defined when either
      ALLOW_ATM_TEMP or ALLOW_DOWNWARD_RADIATION is undef

    * remove trailing blanks
    * remove unused variables
```

Figure: recent `git log` for the MITgcm

# git greatest hits: status

`git status`

```
[tim ctrl]$ git status
On branch ctrl_rec_fix
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working d
irectory)

        modified:   ctrl_get_gen_rec.F

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

        another_file.F

no changes added to commit (use "git add" and/or "git commit -a")
[tim ctrl]$
```

# git greatest hits: diff

```
git diff
git diff <filename>
git diff <commit number> <filename>
```

Compare your changes with past versions. Get commit numbers with `git log <filename>`

```
[tim ctrl]$ git diff ctrl_get_gen_rec.F
diff --git a/pkg/ctrl/ctrl_get_gen_rec.F b/pkg/ctrl/ctrl_get_gen_r
ec.F
index 3e339c115..6444c9f2e 100644
--- a/pkg/ctrl/ctrl_get_gen_rec.F
+++ b/pkg/ctrl/ctrl_get_gen_rec.F
@@ -162,10 +162,8 @@ c     Set switches for reading new records.
         endif
        endif

-c      count0 = fldcount
-c      count1 = fldcount + 1
-       count0 = fldcount - startrec + 1
-       count1 = fldcount - startrec + 2
+       count0 = fldcount
+       count1 = fldcount + 1

        call cal_TimeInterval( fldsecs, 'secs', difftime, mythid )
        call cal_AddTime( fldstartdate, difftime, flddate, mythid
)
```

Figure: `git diff ctrl_get_gen_rec.F`

```
[tim ctrl]$ git diff 7bfe6112e8521d3e9fa523b3deb9ed2c55824670 ctrl_get_gen_rec.F
diff --git a/pkg/ctrl/ctrl_get_gen_rec.F b/pkg/ctrl/ctrl_get_gen_rec.F
index 9b9ac58ee..6444c9f2e 100644
--- a/pkg/ctrl/ctrl_get_gen_rec.F
+++ b/pkg/ctrl/ctrl_get_gen_rec.F
@@ -1,6 +1,3 @@
-C $Header: /u/gcmpack/MITgcm/pkg/ctrl/ctrl_get_gen_rec.F,v 1.15 2012/08/10 19:38:
57 jmc Exp $
-C $Name:  $
-
 #include "CTRL_OPTIONS.h"

       subroutine ctrl_get_gen_rec(
@@ -165,10 +162,8 @@ c     Set switches for reading new records.
         endif
        endif

-c      count0 = fldcount
-c      count1 = fldcount + 1
-       count0 = fldcount - startrec + 1
-       count1 = fldcount - startrec + 2
+       count0 = fldcount
+       count1 = fldcount + 1

        call cal_TimeInterval( fldsecs, 'secs', difftime, mythid )
        call cal_AddTime( fldstartdate, difftime, flddate, mythid )
[tim ctrl]$
```

Figure: diff with a specific commit number for a specific file

# git greatest hits: pull

`git pull`

combines `git fetch` and `git merge`

`git pull --rebase`

`git fetch` + `git rebase`

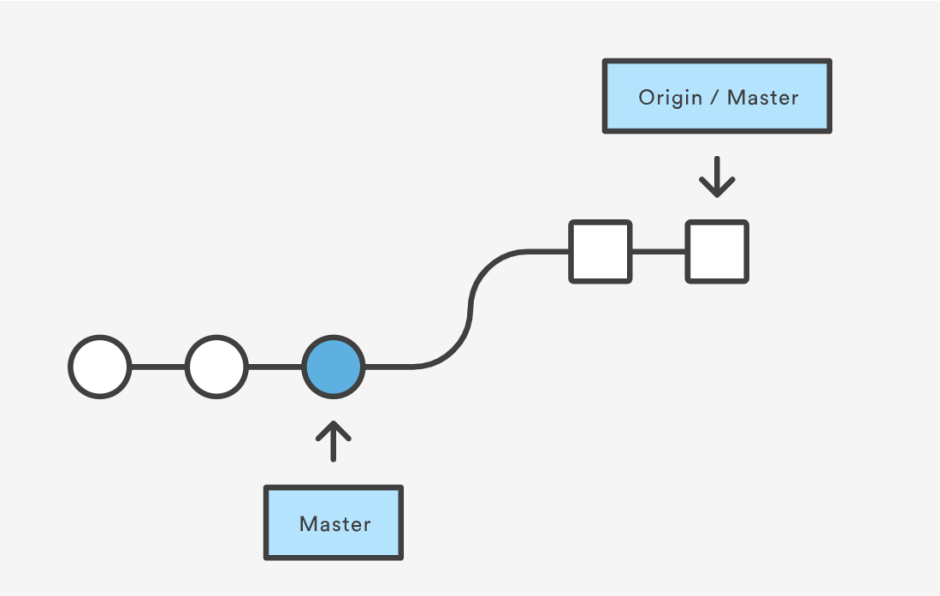



Figure: Repo status prior to running `git pull`, from the Atlassian git tutorial

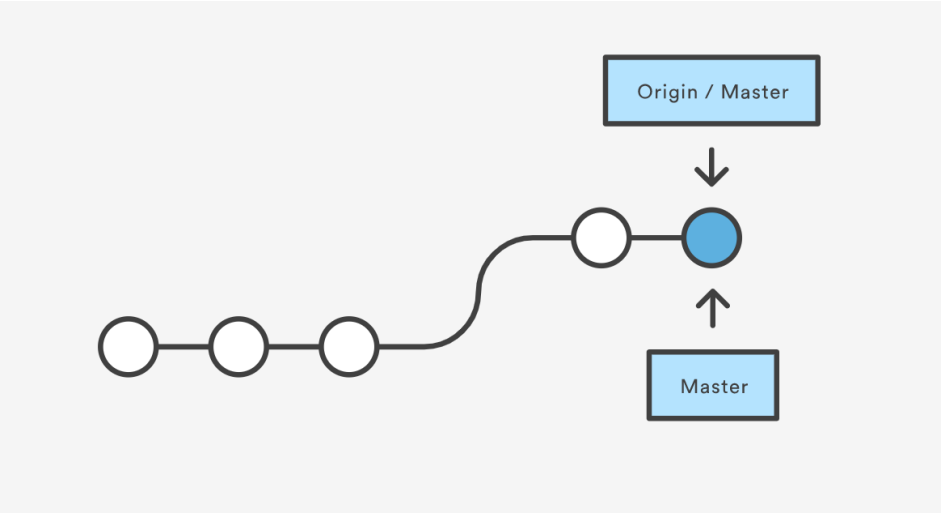



Figure: After `git pull`, from the Atlassian git tutorial

# git greatest hits: add

`git add <filename>`

stage your contribution

```
[tim ctrl]$ git status
On branch ctrl_rec_fix
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working d
irectory)

        modified:   ctrl_get_gen_rec.F

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

        another_file.F

no changes added to commit (use "git add" and/or "git commit -a")
[tim ctrl]$
```

Figure: Repo status after making some changes

```
[tim ctrl]$ git add ctrl_get_gen_rec.F
[tim ctrl]$ git status
On branch ctrl_rec_fix
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   ctrl_get_gen_rec.F

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

        another_file.F

[tim ctrl]$
```

Figure: `git add ctrl_get_gen_rec.F`

# git greatest hits: commit

```
git commit
git commit -m "a nice message"
```

create a logical packet, with a unique ID number and variably helpful message

```
dcount"rl]$ git commit -m "don't subtract startrec from fl
[ctrl_rec_fix ba751988f] don't subtract startrec from fldcount
 1 file changed, 2 insertions(+), 4 deletions(-)
[tim ctrl]$ git log
commit ba751988fbf1eb55c03efe7eb6f99718fa5ceb80 (HEAD -> ctrl_rec_
fix)
Author: Timothy Smith <timsmith204@utexas.edu>
Date:   Mon May 27 13:45:36 2019 -0700

    don't subtract startrec from fldcount

commit 351257375525bb8f9dab9a7ac2318d70e2505f80 (upstream/master,
origin/master, origin/HEAD, master)
Author: Timothy Smith <timsmith204@utexas.edu>
Date:   Mon May 27 11:26:01 2019 -0700

    Added stampede2 optfiles (#246)

    * Added stampede2 optfiles
    * remove trailing blank
    * rename mpi->impi
    * Remove "-DALWAYS_USE_MPI"
      ALWAYS_USE_MPI has been (completely) removed from the code.
```

```
[tim ctrl]$ git status
On branch ctrl_rec_fix
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

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

        another_file.F

nothing added to commit but untracked files present (use "git add"
 to track)
[tim ctrl]$
```

# `git` greatest hits: push

`git push`

"publish" your changes to the cloud

```
[tim ctrl]$ git push --set-upstream origin ctrl_rec_fix
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 465 bytes | 465.00 KiB/s, done.
Total 5 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
remote:
remote: Create a pull request for 'ctrl_rec_fix' on GitHub by visiting:
remote:      https://github.com/timothyas/MITgcm/pull/new/ctrl_rec_fix
remote:
To github.com:timothyas/MITgcm.git
 * [new branch]          ctrl_rec_fix -> ctrl_rec_fix
Branch 'ctrl_rec_fix' set up to track remote branch 'ctrl_rec_fix' from 'origin'.
[tim ctrl]$
```

# The .gitignore file

Create file <repo>/.gitignore to ignore certain file types

```
[tim pych]$ git status
On branch master
Your branch is up to date with 'origin/master'.

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

        file.weird_extension

nothing added to commit but untracked files present (use "git add" to track)
[tim pych]$
```

Figure: 1. suppose we want to ignore all .weird_extension files

```
[tim pych]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   .gitignore

no changes added to commit (use "git add" and/or "git commit -a")
[tim pych]$
```

Figure: 3. now it's gone

vim — bash — tim

```
1  # example
2  *.weird_extension
3
4  # Byte-compiled / optimized / DLL files
5  __pycache__/
6  *.py[cod]
7  *$py.class
8
9  # C extensions
10 *.so
11
12 # Distribution / packaging
13 .Python
14 build/
15 develop-eggs/
16 dist/
17 downloads/
18 eggs/
19 .eggs/
20 lib/
21 lib64/
22 parts/
23 sdist/
```

Figure: 2. add this extension to the .gitignore file

# Clone a repository

Think about this like an "interactive download" - a download where you can always update changes somebody else makes.

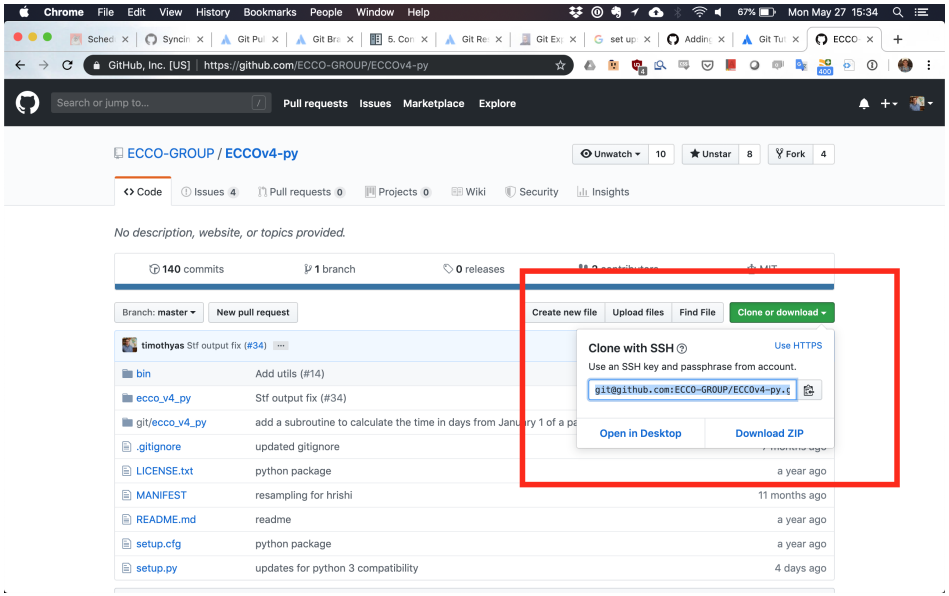



Figure: clone an existing repository

Then run the command

```
git clone <copied-link>
```

wherever you want the directory

# Fork a repository

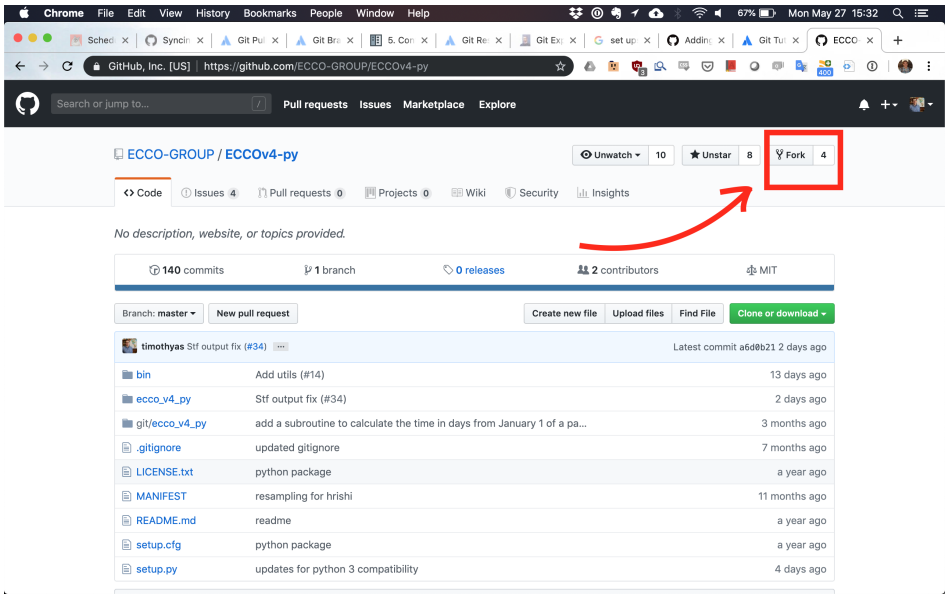



Figure: Fork your favorite repo!

- then clone it to a desired location
- Your URL will look like

  `https://github.com/<username>/<repo-name>.git`
- The upstream URL will look like

  `https://github.com/<group-name>/<repo-name>.git`

# Track upstream changes

- In the terminal, set your branch to track the upstream branch

  ```
  git remote add upstream <upstream-url>
  ```

- So whenever you want to update your local copy with upstream updates:

  ```
  git pull upstream master
  ```

- To submit changes to the upstream code, check out a new branch from your forked repository

  ```
  git checkout -b cool_new_feature
  ```

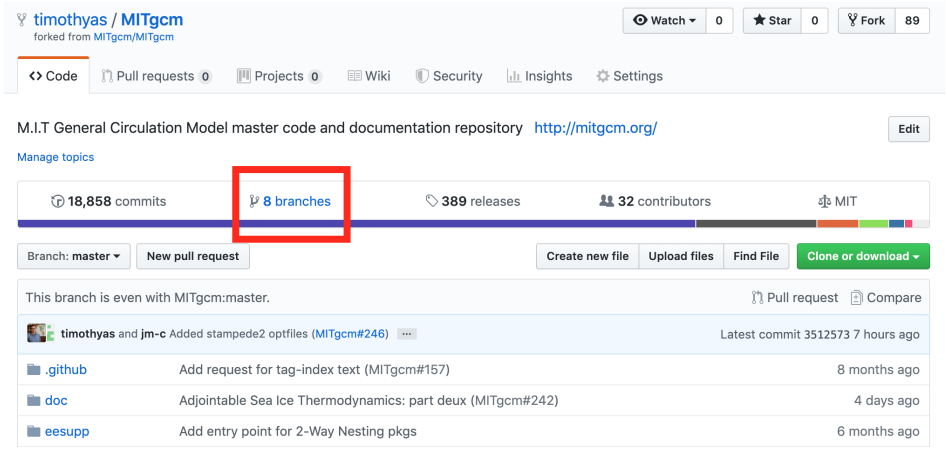
  make some nice edits ...

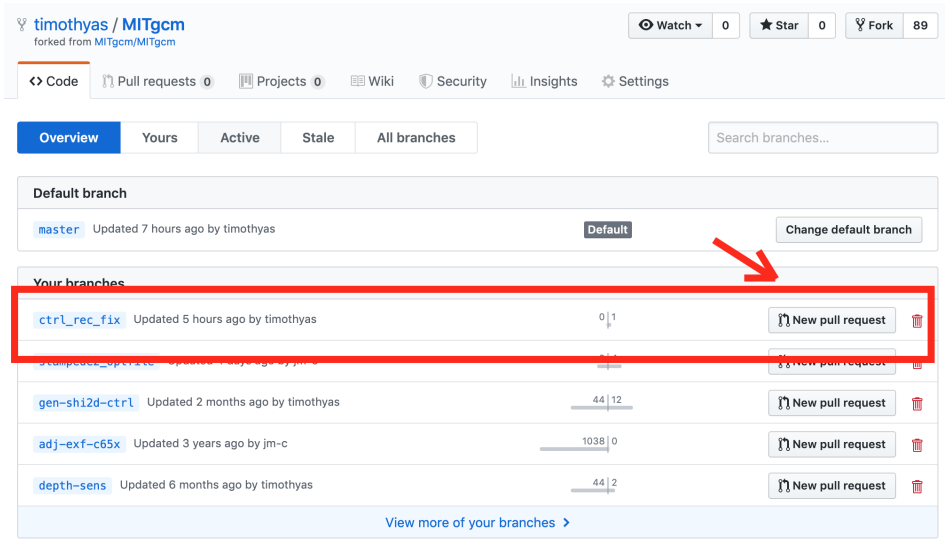
  ```
  git push -u origin cool_new_feature
  ```

- submit by opening a pull request

See the xmitgcm documentation or the MITgcm documentation for a nice reference on git workflow

# Open up a pull request



# Additional references

- understanding reset: how to undo changes and better understand under-the-hood
- Atlassian git tutorial
- MITgcm's git crash course
- apply for student developer pack
- advertised benefits for students
- discounts for students
- apply for educator/reseracher discount
- set up ssh keys for repos