Git and GitHub

CS 4411
Spring 2020
If that doesn’t fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of “It’s really pretty simple, just think of branches as...” and eventually you’ll learn the commands that will fix everything.
Outline for Today

- Git overview
  - Git vs. GitHub
- Basic Git commands
- Conflicts and merges
- Branches
- Recovering from errors
Git Model

Remote repo

Commits

A → B → C

Head

Commit C:

/\src/grass/process.c
/\src/grass/process.h
/\src/apps/mt.c

Makefile

src

|--apps
|--earth
|--grass

|--process.c

|--process.h

|--lib

Local repo

Repo contents

“Working tree”

Origin
Making a Commit

Remote repo

Origin

Local repo

Push

Commit D:
/src/apps/mt.c
/src/apps/myprogram.c
/src/make/Makefile.apps
Outline

• Git overview
  • Git vs. GitHub

• Basic Git commands

• Conflicts and merges

• Branches

• Recovering from errors
Getting Started with Clone

- **git clone**: Create a new local repository by copying a remote repo

  $ git clone https://github.coecis.cornell.edu/cs4411-2020sp/ejt64-egos.git

- **Result**: New folder named “ejt64-egos” in current directory, containing new git repo
  - Contents identical to repo on server
  - **Origin** set to https://github.coecis.cornell.edu/cs4411-2020sp/ejt64-egos.git
HTTPS or SSH?

- Easy to start, no setup required
- Must enter username and password every time you pull or push
- Once set up, no username or password required
- Need to create an SSH key on your computer and add it to your Cornell GitHub account

```bash
git clone https://github.coecis.cornell.edu/cs4411-2020sp/ejt64-egos.git
```

```bash
git clone git@github.coecis.cornell.edu:cs4411-2020sp/ejt64-egos.git
```
Add and Commit

1. Make changes to files
2. Choose some changed files that you’re ready to “publish”
3. git add the changed files
4. git commit and write a message

 ~/egos$ vim src/apps/mt.c
 ~/egos$ vim src/grass/process.c
 ~/egos$ git add src/apps/mt.c
 ~/egos$ git commit

Commit E: /src/apps/mt.c
Pull and Push

- At first, a commit is only on your local repo
- `git push` copies commits to the “origin” remote repo
- `git pull` downloads commits from origin and applies them

```
Local repo
A → B → C → D
```

```
Origin repo
A → B → C → D
```

```
Partner’s repo
A → B → C → D
```

```
```
Understanding Git Status

~/egos$ git status
On branch master
Your branch is up to date with 'origin/master'

Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

modified: src/grass/process.c

Changes not staged for commit:
(use "git add <file>..." to update what will be committed)

modified: src/grass/disksvr.c

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

src/apps/myprogram.c

Current branch
Whether you have unpushed commits
Changes you have added with git add
git knows a file has changed, but you haven’t added it yet
New files you have not yet added in any commit
Git Status After a Commit

~/egos$ git status
On branch master
Your branch is 1 commit behind 'origin/master'

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)

  modified:   src/grass/process.c

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

  src/apps/tags
  src/grass/tags

You made a commit, but haven’t pushed
After your last commit, you continued editing this file
These files still haven’t been added to any commit
Ignoring Files You’ll Never Add

- Some files you never want to commit: ctags files, compiled output, LaTeX aux files...
- Git will keep bothering you about them in git status
- Add a file named .gitignore to the root of your repo, and then add it to a commit

```bash
~/.egos$ cat .gitignore
# ctags files
tags
# LaTeX junk
*.aux
*.log
*.bbl
# The debug log directory
logs/
# Object files in the build dir
build/**/*.o
```

https://www.atlassian.com/git/tutorials/saving-changes/gitignore
Diff: What Am I Committing?

~/egos$ git diff
diff --git a/src/lib/queue.c b/src/lib/queue.c
index c638853..19106c3 100644
--- a/src/lib/queue.c
+++ b/src/lib/queue.c
@@ -19,7 +19,7 @@ struct element {
  void queue_init(struct queue *q){
    q->first = 0;
    q->last = &q->first;
-   q->nelts = 0;
+   q->num_elements = 0;
  }
}

  /* Put it on the wrong side of the queue. I.e., make it the next
@@ -34,7 +34,7 @@ void queue_insert(struct queue *q, void *item){
  }
Diff Details

- Shows differences only for that file

- Why does it give no results? I know I made changes!
- Answer: you have already `git add`ed your changes

```bash
~/egos$ git diff src/grass/process.c

~/egos$ git diff
~/egos$

~/egos$ git diff --staged
diff --git a/src/lib/queue.c b/src/lib/queue.c
index c638853..19106c3 100644
--- a/src/lib/queue.c
+++ b/src/lib/queue.c
@@ -19,7 +19,7 @@ struct element {
```
Un-Adding and Deleting

• Oops, I didn’t mean to add that file!

```bash
~/egos$ git add src/lib/queue.c
~/egos$ git reset HEAD src/lib/queue.c
```

• Telling git you want to delete myprogram.c:

```bash
~/egos$ git rm src/apps/myprogram.c
~/egos$ git status
On branch master
Your branch is up to date with 'origin/master'

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

  deleted: src/apps/myprogram.c
```
Renaming

- Telling git you want to rename myprogram.c:

```
~/egos$ git mv src/apps/myprogram.c src/apps/newname.c
~/egos$ git status
On branch master
Your branch is up to date with 'origin/master'

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed: src/apps/myprogram.c -> src/apps/newname.c
```

- Otherwise, git will think you deleted myprogram.c, and both myprogram.c and newname.c will end up in the repo.
Outline

• Git overview
  • Git vs. GitHub
• Basic Git commands
• Conflicts and merges
• Branches
• Recovering from errors
Concurrent Changes

GitHub repo

My local repo

Partner’s local repo

A → B → C → D

A → B → C → E
Possible Outcomes

- No conflicts, just merge

```
~/egos$ git pull
# Editor pops up
Merge made by the 'recursive' strategy
  src/lib/queue.c
  1 file changed, 7 insertions (+), 7 deletions (-)
```

Partner’s local repo

```
Merge branch 'master' of https://github.coecis.cornell.edu/etremel/egos.git
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit
```
Possible Outcomes

• Conflicting changes to the same file(s)

```bash
~/egos$ git pull
Auto-merging src/lib/queue.c
CONFLICT (content): Merge conflict in src/lib/queue.c
Automatic merge failed; fix conflicts and then commit the result
~/egos$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 1 and 1 different commits each, respectively.
...
Unmerged paths:
  (use "git add <file>..." to mark resolution)

  both modified: src/lib/queue.c
```
Conflict File Syntax

Inside queue.c:

```c
void queue_add(struct queue *queue, void *item){
    struct element *e = calloc(1, sizeof(*e));
    e->item = item;
    e->next = 0;
    *queue->last = e;
    queue->last = &e->next;
    queue->nelts++;
}
```

Your local version of conflicting lines

```c
<<<<< HEAD
*queue->last = e;
queue->last = &e->next;
queue->nelts++;
=======
```

HEAD means these changes are in your local HEAD commit

```c
*q->last = e;
q->last = &e->next;
q->num_elements++;
```

Origin repo’s version of conflicting lines

Hash of commit from origin that contains these conflicting changes

354a72479204de581ffa83551843b92e585506b8
Finishing the Merge

• Edit the file to choose a single version of the conflicting lines
• Make sure to delete the <<<<<<<<<<< and ======= lines!
• When you have resolved the conflict:

```bash
~/egos$ git add src/lib/queue.c
~/egos$ git commit
# Write a message for the merge commit
~/egos$ git push
```
Outline

• Git overview
  • Git vs. GitHub
• Basic Git commands
• Conflicts and merges
• Branches
• Recovering from errors
Git Branches

- Track different sequences of commits diverging from common starting point
- Make explicit what happened already when you & your partner made conflicting commits
- Let you choose when to merge
Branch Basics

• Creates a new branch, pointing to same commit as master

• New commit goes on thread-develop, master still points to last commit
Branch Basics

- Changes your repo’s active branch to master – you’ll notice the changes you made to mt.c are gone

```bash
~/egos$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'
```

- This commit goes on master, and the HEAD pointer for master moves up

```bash
~/egos$ git add src/grass/process.c
~/egos$ git commit
```
Pushing and Pulling

- You want to push the commit on your new branch to GitHub:
  ```
  ~/egos$ git checkout thread-develop
  Switched to branch 'thread-develop'
  ~/egos$ git push
  fatal: The current branch thread-develop has no upstream branch.
  To push the current branch and set the remote as upstream, use
  git push --set-upstream origin thread-develop
  ```
- OK, do what it says
- After this point, `git push` will just work
Why Was That Necessary?

- checkout -b only creates a branch on your local repo
- GitHub repo won’t add a commit to a branch that doesn’t exist
Pushing and Pulling

- You want to switch to a new branch your partner created

```
~/egos$ git checkout thread-develop
error: pathspec 'thread-develop' did not match any file(s) known to git
```

- Why didn’t that work?

```
~/egos$ git pull
Remote: Enumerating objects
...
From github.coecis.cornell.edu:etremel/egos
* [new branch]   thread-develop -> origin/thread-develop
Already up to date.
~/egos$ git checkout thread-develop
Branch 'thread-develop' set up to track remote branch 'thread-develop' from 'origin'
Switched to a new branch 'thread-develop'
```
Merging Branches

- Eventually, you’ll want to merge your branch back into master
- First, switch your current branch to **master**
  ```
  ~/egos$ git pull
  ~/egos$ git checkout master
  ```
- Then, merge the feature branch **into** master
  ```
  ~/egos$ git merge new-feature
  ~/egos$ git push
  ```

Resolve any merge conflicts, same as when you pull and see conflicts

Publish the merge commit, so everyone else can see the merge
Repository Design with Branches

- In many software teams, **master** is the “stable” branch
- Each new feature or bug fix is developed on its own branch
- When tested and safe, branch is merged into master
- While developing on a branch, merge from **master** into your branch to get updates and bugfixes
Outline

• Git overview
  • Git vs. GitHub
• Basic Git commands
• Conflicts and merges
• Branches
• Recovering from errors
**Some Useful Incantations**

- **Oops! I made a commit but then made one small change**

  ```
  # Do this BEFORE you push
  $ git add changed-file.c
  $ git commit --amend --no-edit
  ```

- **Oops! I want to edit the message on the commit I just made**

  ```
  $ git commit --amend
  # Edit the message in your editor
  ```

- **Oops! I deleted a file and didn’t mean to**

  ```
  $ git checkout path/to/file.c
  ```
Branch-Related Problems

• I just made a commit to master, but I should have put it on a new branch instead

# Create a new branch with the same state as master
$ git branch new-branch-name
# Remove the latest commit from master
$ git reset HEAD~ --hard
# Switch to the new branch, which still has the commit
$ git checkout new-branch-name
Branch-Related Problems

• I just committed to the wrong branch – I thought I was on the queue-develop branch but I’m on the experiments branch

```bash
# Undo the last commit, but leave files changed
$ git reset HEAD~ --soft

# Stash the changed files, then move to the right branch
$ git stash

$ git checkout queue-develop

$ git stash pop

# Commit the changes on the correct branch
$ git add queue.c queue.h

$ git commit
```
The Unnecessary Merge

• Make changes, commit, push... then realize your repo is stale

```bash
$ git push
! [rejected] master -> master (fetch first)
error: failed to push some refs to 'https://github.com/etremel/egos.git'
hint: Updates were rejected because the remote contains work that you do
hint: not have locally. This is usually caused by another repository pushing
hint: to the same ref. You may want to first integrate the remote changes
hint: (e.g., 'git pull ...') before pushing again.
```

• Upon pull, you’re prompted to create a merge commit

```bash
$ git pull
```

Merge branch 'master' of https://github.com/etremel/egos.git

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit
The Unnecessary Merge

• If I had known to pull first, I wouldn’t have to merge!

• First, ensure the merge aborts without making a merge commit
  • Either due to genuine conflict, or by deleting the merge commit message and saving (empty commit message aborts the commit)

• Then:

```
$ git reset --hard
$ git reset HEAD^  # Discard changes made by merge
$ git stash          # Undo your last commit, leaving files changed
$ git pull # no merge this time  # Stash your changes, then pull again
$ git stash pop
$ git add foo.c bar.c # etc
$ git commit         # Re-do your commit on the new head
```
Further Reading

• Atlassian Git Tutorials:
  https://www.atlassian.com/git/tutorials

• Detailed documentation on every command:
  https://git-scm.com/docs

• Happy Git with R’s “Useful Git Patterns:”
  https://happygitwithr.com/workflows-intro.html

• Oh Shit, Git!?! (source of my error-recovery examples):
  https://ohshitgit.com/