Introduction to Version Control

- Conflict resolution

What about "conflicts"?

- Suppose you've fetched changes and merged them into your main branch
- Normally we see a clean report of git's work
  - Downloading objects from remote
  - Merging changes into our working copy (i.e., new files, removed files, edits to existing files)

```
$ git pull
<password>
remote: Counting objects: 19, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 18 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (18/18), done.
From ssh://git.example.com/repo/calc
   b5b22e2..4955e13  main     -> origin/main
Updating b5b22e2..4955e13
Fast-forward
 calc/README.md      | 4 ++++
 calc/main_init.py   | 3 +++
 calc/guiframe.py    | 1 +
 3 files changed, 8 insertions(+)
 create mode 100644 calc/README.md
 create mode 100644 calc/main_init.py
 create mode 100644 calc/guiframe.py
```

What about "conflicts"?

- However, sometimes we don't see a clean report

What about "conflicts"?

- However, sometimes we don't see a clean report
- Scenario:
  - Meg has pulled from the remote. Both she and Stevie have the first version of button.c shown here.

```
/* button.c */

#include <stdio.h>
#include <gui.h>
```

What about "conflicts"?

- However, sometimes we don't see a clean report
- Scenario:
  - Meg has pulled from the remote. Both she and Stevie have the first version of button.c shown here.
  - Stevie makes a change to calc/button.c, **but doesn't commit**

```
/* button.c */

#include <stdio.h>
#include <gui.h>
```

```
/* button.c */

#include <stdio.h>
#include <gui.h>

/* I, Stevie, am the smartest Griffin */
int _init_button (int id, int skin,
    char *label) { /* A stub, I say! */ }
```

What about "conflicts"?

- However, sometimes we don't see a clean report
- Scenario:
  - Meg has pulled from the remote. Both she and Stevie have the first version of button.c shown here.
  - Stevie makes a change to calc/button.c, **but doesn't commit**
  - Meg makes a change to calc/button.c, **then commits and pushes it**.

```
/* button.c */

#include <stdio.h>
#include <gui.h>
```

```
/* button.c */

#include <stdio.h>
#include <gui.h>

/* I, Stevie, am the smartest Griffin */
int _init_button (int id, int skin,
    char *label) { /* A stub, I say! */ }
```

```
/* button.c */

#include <stdio.h>
#include <gui.h>

void _init_button (int code, int look,
    char *title) { /* For you, Connie! */ }
```

What about "conflicts"?

- However, sometimes we don't see a clean report
- Scenario:
  - Meg has pulled from the remote. Both she and Stewie have the first version of button.c shown here.
  - Stewie makes a change to calc/button.c, **but doesn't commit**
  - Meg makes a change to calc/button.c, **then commits and pushes it**.
  - Afterwards Stewie tries to commit and then push

```
/* button.c */

#include <stdio.h>
#include <gui.h>
```

```
/* button.c */

#include <stdio.h>
#include <gui.h>

/* I, Stevie, am the smartest Griffin */
int _init_button (int id, int skin,
    char *label) { /* A stub, I say! */ }
```

```
/* button.c */

#include <stdio.h>
#include <gui.h>

void _init_button (int code, int look,
    char *title) { /* For you, Connie! */ }
```

What about "conflicts"?

```
meg$ git add button.c
meg$ git commit -m 'That will be a swell looking button!'
[main bda4b22] That will be a swell looking button!
 1 file changed, 4 insertions(+)
meg$ git push
<password>
<... snip ...>
Total 4 (delta 1), reused 0 (delta 0)
To ssh://stewie@git.example.com/repo/calc
   15c486a..bda4b22  main -> main
```

```
stewie$ $ git commit -a -m "A sterling job, I say, with _init_button."
[main 6708583] A sterling job, I say, with _init_button.
 1 file changed, 4 insertions(+)
stewie$ git push
<password>
<... snip ...>
error: failed to push some refs to 'ssh://stewie@git.example.com/repo/calc'
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.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
stewie$ curses!
-bash: curses!: command not found
```

What about "conflicts"?

stewie$ $ git pull
<password>
remote: Counting objects: 7, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (4/4), done.
From ssh://git.example.com/repo/calc
   15c486a..bda4b22 main   -> origin/main
Auto-merging calc/button.c
CONFLICT (content): Merge conflict in calc/button.c
Automatic merge failed; fix conflicts and then commit the result.

stewie$ git status
-On branch main
Your branch and 'origin/main' have diverged,
and have 1 and 1 different commit each, respectively.
  (use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")

Unmerged paths:
  (use "git add <file>..." to mark resolution)

                both modified: button.c

no changes added to commit (use "git add" and/or "git commit -a")

Resolving a merge conflict (what Stewie sees)

```
/* button.c */

#include <stdio.h>
#include <gui.h>

<<<<<<< HEAD
/* I, Stevie, am the smartest Griffin */
int _init_button (int id, int skin,
    char *label) { /* A stub, I say! */ }
=======
void _init_button (int code, int look,
     char *title) { /* For you, Connie! */ }

>>>>>>> bda4b226dc91226a0ab310ad4a7feef2c069b4b4
```

- git indicates the conflict with a bit of markup
  - "<<<<<<< HEAD" to "=======": Stevie's original code
  - "=======" to ">>>>>>> bda4b22…": What Meg pushed

Resolving a merge conflict

- Before Stewie can successfully commit and push his work to the remote, he must resolve the conflict
- There are two parts to this:
  - The **human part** (harder)
  - The **technical part** (easy)
- The human part is to decide what code in button.c should be kept
- The technical part is editing the file in a way reflecting that decision, then commiting (and possibly pushing)

Resolving a merge conflict

```
/* button.c */

#include <stdio.h>
#include <gui.h>

<<<<<<< HEAD
/* I, Stevie, am the smartest Griffin */
int _init_button (int id, int skin,
    char *label) { /* A stub, I say! */ }
=======
void _init_button (int code, int look,
     char *title) { /* For you, Connie! */ }

>>>>>>> bda4b226dc91226a0ab310ad4a7feef2c069b4b4
```

before

```
/* button.c */

#include <stdio.h>
#include <gui.h>

int _init_button (int id, int skin,
    char *title) { /* This stub is to be completed */ }
```

after

What about "conflicts"?

stewie$ $ vim calc/button.c
<actions to edit the file>

stewie$ git commit –a –m "button.c now back on track"
[main 8d0c566] button.c now back on track

stewie$ git push
<password>
<... snip ...>
Counting objects: 14, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (8/8), done.
Writing objects: 100% (8/8), 1.04 KiB | 0 bytes/s, done.
Total 8 (delta 2), reused 0 (delta 0)
To ssh://stewie@git.example.com/repo/calc
   bda4b22..8d0c566 main -> main

meg$ git pull
<password>
<... snip ...>
Unpacking objects: 100% (8/8), done.
From ssh://git.example.com/repo/calc
   bda4b22..8d0c566 main   -> origin/main
Updating bda4b22..8d0c566
Fast-forward
 calc/button.c | 5 ++---
 1 file changed, 2 insertions(+), 3 deletions(-)

git commands

- Note:
  - We've already referred to git "commands"
  - Yet git itself **is a UNIX command**
- A git command is how we specify an action from the git client
- Syntax
  - git command [option] [arguments]
- The number of git commands and options is very large
  - We'll be focusing on a much smaller subset of these.
  - (Beware of Google and StackOverflow as answers there can lead you astray…)

Previously seen

```
$ git clone    ssh://stewie@git.example.com/repo/calc

$ git add      button.c

$ git fetch
```

Git command

argument

A few more examples

```
$ git remote    add origin https://git.420.com/fubar

$ git commit   -a –m "Fixed typo in label"

$ git diff     --name-status HEAD
```

Git command options & arguments

Some useful commands

clone	make a local copy of a remote repo
add	stage files/directories so they'll be ready to be committed
commit	store into local repo a snapshot of working-copy changes
status	list working copy files/dirs differing from local repo
log	output commit messages with their snapshot SHA-1 hashes
diff	show differences of working-dir contents with local repo
pull	fetch and merge into local repo any remote repo changes
push	transfer local repo snapshots to remote repo
fetch	download data for remote changes to local repo (but no more!)
merge	combine new remote changes with files in local repo
rm	remove file from working copy / working tree
init	convert working directory into git local repo (caution!)

If you need help…

- For a specific command:
  - git <command> --help
  - Provides list of arguments and options
- For info on repository access
  - speak to the provider, or
  - read the provided documentation
  - follow the instructions given in the lab and on the assignments!