Before we begin, update your VM...

Ensure all local packages are up to date

- **FIRST SET A CHECKPOINT FOR YOUR VM**
- `sudo apt-get update`

Upgrade/Install some packages

- `sudo apt-get upgrade -y gdb`
- `sudo apt-get install -y python3-pip`
- `pip3 install --upgrade pip`

Tips and Tricks Update

- Edit your `~/.ssh/config` to contain the following (works on MacOS, Linux, VM, and WSL)

```
### CAEN
Host caen login.engin.umich.edu
  HostName login.engin.umich.edu
  User mmdarden  # Use your own uniqname
  ControlMaster auto
  ControlPath ~/.ssh/_%r@%h:%p
  ControlPersist 43200
```

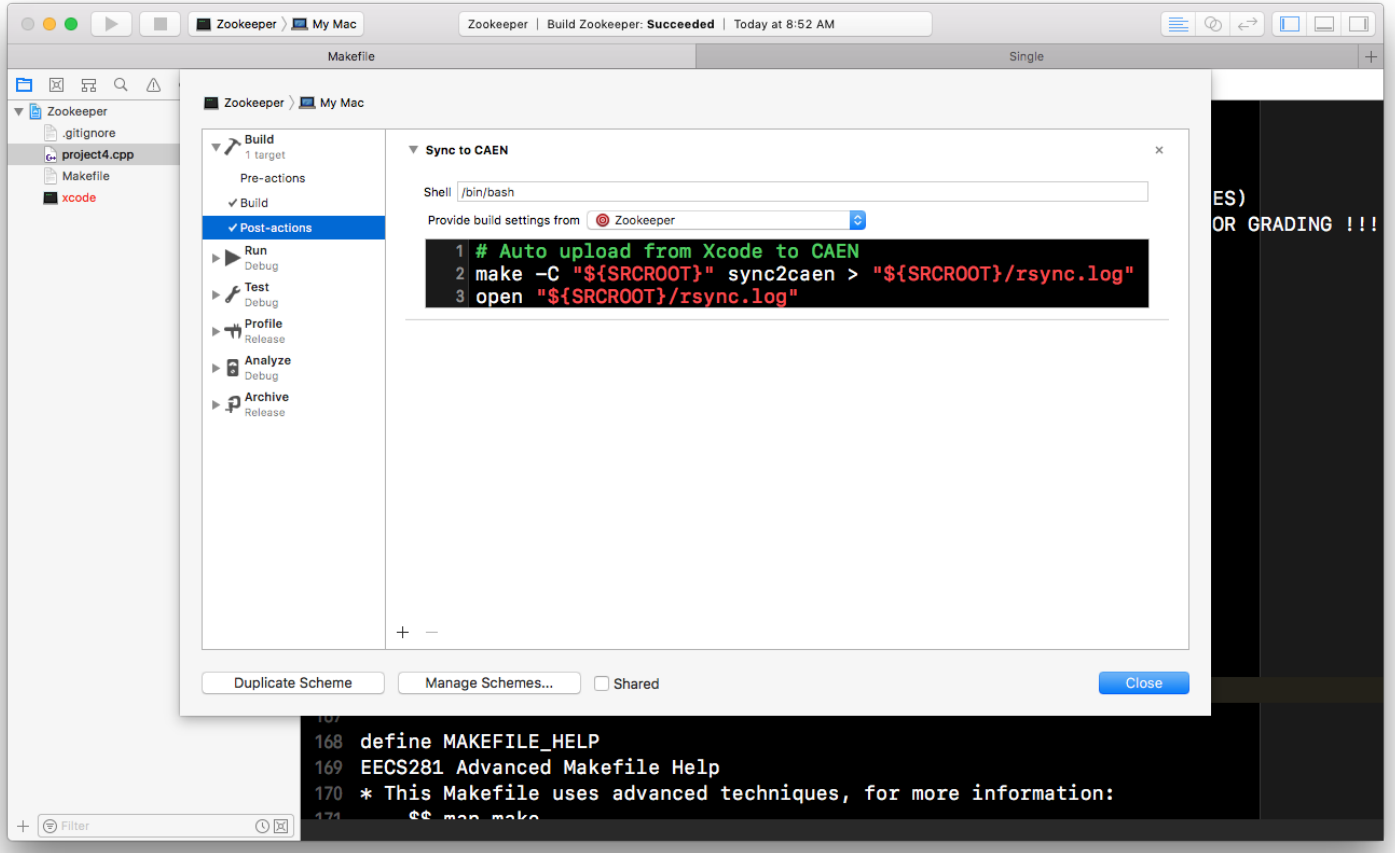
- When connecting to CAEN (with `ssh caen`)
  - First login requires password and DUO
  - Subsequent logins connect instantly (for 12 hours, or until…)
  - When the multiplexing expires or is broken (rules unknown)
  - Works for everything that uses ssh (commands, sessions, 3rd party software, etc.)
  - 2 useful commands
    - `ssh -O check caen`
    - `ssh -O stop caen`
  - Also, look for the file `~/.ssh/_mmdarden@login.eecs.umich.edu:22`

TTU++

- Connect your local dev environment to CAEN
  - Use `rsync` and a "Post-build script"
- EECS 281 example: [https://gitlab.eecs.umich.edu/eecs281/makefile]
  - Look at `$(REMOTE_BASEDIR)`
  - Look at `$(REMOTE_PATH)`
  - Look at target `sync2caen`
- Xcode example:
  - Edit Scheme…
  - Add a "Build Post-action"
  - Name: "Sync to CAEN"
  - Shell: `/bin/bash`
  - Provide build settings from: `<current scheme>`
  - Add the following script

```
# Auto upload from Xcode to CAEN
make -C "${SRCROOT}" sync2caen > "${SRCROOT}/rsync.log"
open "${SRCROOT}/rsync.log"
```

- Check on CAEN in `~/$(REMOTE_PATH)`
- Sync happens after every successful build!


Zookeeper
My Mac
Zookeeper | Build Zookeeper: Succeeded | Today at 8:52 AM
Makefile
Single
Zookeeper
.gitignore
project4.cpp
Makefile
xcode
Build
1 target
Pre-actions
Build
Post-actions
Run
Debug
Test
Debug
Profile
Release
Analyze
Debug
Archive
Release
Sync to CAEN
Shell /bin/bash
Provide build settings from Zookeeper
Auto upload from Xcode to CAEN
make -C "${SRCROOT}" sync2caen > "${SRCROOT}/rsync.log"
open "${SRCROOT}/rsync.log"
Duplicate Scheme
Manage Schemes...
Shared
Close
ES)
OR GRADING !!!
168 define MAKEFILE_HELP
169 EECS281 Advanced Makefile Help
170 * This Makefile uses advanced techniques, for more information:

Debuggers

What Does gdb Do?

Yes

- Start your program (with options and arguments)
- Stop your program
- Allow you to see into registers and memory
- Allow you to change values manually during execution

What Does gdb Do?

Yes

- Start your program (with options and arguments)
- Stop your program
- Allow you to see into registers and memory
- Allow you to change values manually during execution

No

- **MAGIC**

How Do I gdb?

To debug a program with gdb, simply put `gdb` in front of the program, i.e.:

How Do I gdb?

To debug a program with gdb, simply put `gdb` in front of the program, i.e.:

```
> ./prime # running normally
> gdb ./prime # debugging the program with gdb
```

How Do I gdb?

To debug a program with gdb, simply put `gdb` in front of the program, i.e.:

```
> ./prime # running normally
> gdb ./prime # debugging the program with gdb
```

One annoying gotcha shows up if the program to debug takes any options. The simple prime program does not, but if it did:

```
> ./prime --imaginary-option # running normally
> gdb ./prime --imaginary-option # will not work
gdb: unrecognized option '--imaginary-option'
> gdb --args ./prime --imaginary-option # gdb will ignore everything after --a
rgs
```

GDB's Text User Interface

- It's a CLI program, get over it!
- Nope... Beast Mode... GDB TUI
  - At launch with --tui
  - After launch with `C-x 1`

GDB's Text User Interface

- It's a CLI program, get over it!
- Nope... Beast Mode... GDB TUI
  - At launch with --tui
  - After launch with `C-x 1`

GDB TUI Key Bindings (partial)

Binding	**Action**
`C-x a`	Enter/exit TUI
`C-x 1`	Change TUI layout?
`C-x 2`	Change TUI layout
`C-x o`	Switch window focus
`C-x s`	Single Key mode
`C-l`	Refresh screen
`C-p`, `C-n`, `C-b`, `C-f`	Readline navigation (Emacs FTW!)
---	---

GDB TUI Single Key Mode

- This is truly GDB Beast Mode... on steroids!

Key	Action
c	continue
d	down
f	finish
n	next
q	exit the Single Key mode
r	run
s	step
u	up
v	info locals
w	where
---	----------

gdb Commands

`run`

- Starting gdb will not run your program by default. You must use the `run` command to begin execution.
- Using `run` will start your program with the options originally specified, or you can pass new options with `run`.

```
(gdb) run --different-option
```

- If your project is recompiled, each `run` will automatically reload the new version. Debugging is easier if you don't quit gdb, but leave it running in a separate terminal.

gdb Commands

`backtrace`, `up`, `down`, `frame`, `print`

- While your program is running, it has a function call stack that is built up with frames that hold parameters, locals, and register information for each invocation. Consider math.c:

```
#include <iostream>
using namespace std;
int subtract (int a, int b) { return a - b; }
int divide (int a, int *b) { return a / *b; }
int do_math (int x, int y, int z) {
    int temp = subtract(x, y);
    temp = divide(z, &temp);
    return temp;
}
int main () {
    int temp = do_math(10, 10, 20);
    cout << "Result: " << temp << endl;
    return 0;
}
```

Function call stack (growing to the right)

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `finish`, `set`

- Look at your source with `list` or `list <function>`

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `finish`, `set`

- Look at your source with `list` or `list <function>`
- Stop and start your program with `break` and `continue`

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `finish`, `set`

- Look at your source with `list` or `list <function>`
- Stop and start your program with `break` and `continue`
- Take things at your own pace with `step` (into), `next`, and `finish` (out)

gdb Commands

`list`, `break`, `continue`, `step`, `next`, `finish`, `set`

- Look at your source with `list` or `list <function>`
- Stop and start your program with `break` and `continue`
- Take things at your own pace with `step` (into), `next`, and `finish` (out)
- Make a change to variables and registers with `set`

More on breakpoints

- Generally specified by filename:linenumber
- Will also work in context
- List all current breakpoints with `info breakpoints`
- Remove with `delete <number>` or `disable <number>` until later
- Skip over working code with breakpoints on either side and `continue`

Conditional breakpoints

- Unbelievably efficient for debugging
- Can create with `break myfile:11 if x == 5`
- Can extend with `condition <breakpoint number> x == 5`

GDB Does Python!!

- Access to GDB internals
- Variables, functions, etc.
- Inline, short entry, and script
- A pretty printer

```
class ObjectPrinter:
    '''Pretty print an Object'''

    def __init__(self, val):
        self.val = val

    def to_string(self):
        '''Change this to reflect real properties from the object'''
        return self.val

    def display_hint(self):
        return 'Object'

def lookup_type(val):
    return ObjectPrinter(val) if val.type == 'Object' else None

gdb.pretty_printers.append(lookup_type)
```

The New Hotness... gdbgui

- `pip3 install gdbgui --upgrade`
- Rerun the previous debug session
- Start a new debug session

Open Problems with Debugging

Look at `inf.c`

```c
#include <stdio.h>

int recurse(int add_me) {
    if (add_me == 1) {
        return add_me;
    }
    return recurse(add_me + add_me);
}

int main() {
    printf("%d\n", recurse(2));
}
```