Installation + Setup:

- Python Setup 3.9 currently works for docking (if needed, setup a conda environment with python 3.9 version specified):
    - pip install scipy and numpy
    - Blastermaster will fail if you don't have the proper dependencies and will probably fail if you don't have a compatible python version

- Install dock from http://dock.compbio.ucsf.edu/
    - If you have access to **`umms-maom/turbo`** directory, you can use DOCK from there so you won't need to install it

- Follow the instructions here https://github.com/momeara/dock_campaign_template to clone the dock campaign template (usually in **`/your_home_dir/opt`**) and then set up a working directory for your dock campaign
    - After the working directory is setup, follow the instructions to initialize the shell environment in working directory

- Once the environment is set up, edit the script **`setup_dock_environment.sh`**:
    - Fields highlighted in yellow are ones that you will need to fill in but they should be the same as listed here. Fields in red are ones that are personalized to your setup and file paths. The export SLURM lines will need to be uncommented
    - Make sure your scratch directory actually exists
    - This script sets up all the paths and slurm stuff needed to run the docking scripts

```
# supported cluster types:
# LOCAL, SGE, SLURM
export USE_SLURM=true
export CLUSTER_TYPE=SLURM

# use these options for SLURM clusters
export SLURM_ACCOUNT=maom0
export SLURM_MAIL_USER=email
export SLURM_MAIL_TYPE=BEGIN,END
export SLURM_PARTITION=standard

export SCRATCH_DIR=/scratch/maom_root/maom0/scratch_directory
export DOCK_TEMPLATE=path_to/dock_campaign_template

export DOCKBASE=${HOME}/turbo/opt/DOCK-dev/ucsfdock #(this is the current version I have
working)

# the DOCKBASE folder should contain files and folders that look like this:
# analysis, bin, common, docking, install, ligand protein, files.py, util.py, __init__.py

export PATH="${PATH}:${DOCKBASE}/bin:${DOCK_TEMPLATE}/scripts"
export ZINC3D_PATH=" ${HOME}/turbo/ZINC_mirror/published/3D" #(this is the current path to the
zinc database files)
```

```
# to build molecules for zinc (WIP)
#https://comp.chem.umn.edu/sds/amsol/amsol7.1.tar.xz
#with co-linear patch
export AMSOLEXE=${HOME}/opt/amsol7.1-colinear-fix/amsol7.1
export EMBED_PROTOMERS_3D_EXE=$DOCKBASE/ligand/3D/embed3d_corina.sh

export OMEGA_ENERGY_WINDOW=12
export OMEGA_MAX_CONFS=600
```

- Run `source setup_dock_environment.sh` to setup the docking environment. **Note:** you will have to run this every time you open a new shell for docking (you will get an error if you don't)

DOCKing:

Preparing the database:

- To dock, you will need a `database.sdi` file containing db2.gz file paths on each line. These files contain the ligands to be docked to your receptor. To get this `database.sdi` file there are two different docking situations (retrospective and prospective) requiring different steps

- **Retrospective:** Collect db2 ligand files by first collecting a list of active smiles strings (from pubchem or similar databases).
    - Go to https://tldr.docking.org/ and generate actives and decoys db2 files from your file containing the active molecules smiles strings (one string per line)
    - Download the db2.gz files and put them somewhere where you can access them from your docking directory

- **Prospective:** There are already sdi files containing db2.gz files of ligands from the ZINC database in `/nfs/turbo/umms-maom/ZINC_mirror/tranches`
    - If you use one of the sdi files from this directory, copy the file to a directory in your databases directory of your dock campaign directory. I like to name the sdi file `database.sdi` because it matches the name that DOCK looks for when docking

- If the `database.sdi` file does not already exist (usually the case when you set up your own database for retrospective docking), set up a `database.sdi` file in the databases directory containing paths to the db2.gz ligand db2 files by:
    - The `database.sdi` file can be written with the following command
        - `ls -d path_to_db2_files/*.db2.gz > database.sdi`
        - You should put this database.sdi file in a descriptively named directory within the databases directory of your dock campaign working directory (e.g. `path_to_dock_dir/databases/ZINC_instock/database.sdi`)

Making the structure:

- Once you have a database of ligands set up, run `$DOCK_TEMPLATE/scripts/wizard.sh` (the wizard script always needs to be run from your home dock directory) in your dock campaign directory
    - When prompted, choose the ‘Make Structure’ option then choose TEMPLATE_CUSTOM_PDB option and name the structure. **Note:** the date will automatically be added to whatever name you give)

- After this step, a directory in the `structures` directory is created with the name you gave. This directory contains a file `1_make_structure.sh` which you will need to edit

    - If the pdb is in the Protein DataBank, fill in the lines highlighted in yellow
    - If the pdb is not in the Protein DataBank (e.g. designed) or if you already have the structure in the directory, you can comment out the starred lines (****) and add the line highlighted in red. This creates a link from the structure to raw.pdb

```
# fill in
PDB_CODE= ****
RECEPTOR_CHAIN=
LIGAND_CHAIN=
LIGAND_RESID=

wget https://files.rcsb.org/download/${PDB_CODE}.pdb ****
mv ${PDB_CODE}.pdb raw.pdb ****

ln -s pdb_name.pdb raw.pdb

grep "^ATOM................${RECEPTOR_CHAIN}" raw.pdb > rec.pdb
grep "^HETATM..........${LIGAND_RESID} ${LIGAND_CHAIN}" raw.pdb > xtal-lig.pdb

echo "N atoms in rec.pdb: $(wc -l < rec.pdb)"
echo "N atoms in xtal-lig.pdb: $(wc -l < xtal-lig.pdb)"
```

- Once you edit this file, run `source 1_make_structure.sh`
    - This will output a pdb file containing the receptor atoms (`rec.pdb`) and a pdb file containing the ligand atoms (`xtal-lig.pdb`)
    - Make sure to check that the ligand and receptor pdb files are not empty
    - **Note:** if your protein does not have an endogenous ligand, you can predict the binding site using DiffDock

Preparing the structure:

- Return back to the main directory of your dock campaign and run the `wizard.sh` script again (`$DOCK_TEMPLATE/scripts/wizard.sh`) but this time select the ‘Prepare structure’ option → choose the structure you’d like to prepare (the one you just made using the ‘Make structure’ option)

- Choose the TEMPLATE_STANDARD option (if you have the `rec.pdb` and `xtal-lig.pdb` already generated from previous step)
    - This sets up a directory in the `prepared_structures` directory with a file called `1_prepare_structure.sh`
    - cd into this directory and edit the `1_prepare_structure.sh` file (this file shouldn't really need editing if the previous steps have worked correctly)
    - Run the script by `source 1_prepare_structure.sh` and make sure that the output states that the job is submitted. Check to see if blastermaster is running by using the command `sq`
    - Blastermaster creates the grids (electrostatic, van der waals, and desolvation grids) and spheres (receptor and ligand spheres) that will be used for scoring and docking poses during the docking run

- Once blastermaster has finished, make sure a `dockfiles` directory is generated in the directory containing the `1_prep_structure.sh` file and that `dockfiles` contains a file called `INDOCK`. If the `dockfiles` directory or `INDOCK` file does not exist, look at the error logs in the `working` directory to determine the error

- Open the `INDOCK` file and edit as needed – this is the file that controls parameters for docking (like max energy, max ligand size, etc.)
    - To ensure that all ligands are docked for prospective docking, increase the **bump_maximum, bump_rigid, and mol2_score_maximum**, as well as the **atom_maximum** (usually I set these four values to 100)
    - There are other parameters that can be changed but these are the ones that I typically change

Doing the docking:

- If blastermaster ran correctly and the `INDOCK` file is edited as necessary, `cd` back into the main working directory of you dock campaign and run the wizard script again (`$DOCK_TEMPLATE/scripts/wizard.sh`) → choose the 'Docking run' option → choose the database you would like to use → select your prepared structure → choose the TEMPLATE_STANDARD template → name the docking run (default format is nice so you can leave this blank if you want)

- This creates a directory in the `docking_runs` directory named by the docking run tag which contains a file called `1_submit.sh`
    - `cd` into this directory, edit the file, then run it by source `1_submit.sh`. This should start the docking run (check `sq` to make sure it is running) – this will probably fail at first, but follow the steps below for fixing the errors

When running dock, the docking will originally fail because of a path error in your scratch directory. To fix this, cd into your scratch directory and mkdir `DOCK_common`

EDITING DOCK_SUBMIT.SH:

NOTE: when running `1_submit.sh`, some of the paths are not correct in the `dock_submit.sh` file.

- To fix this, make a copy of `${DOCK_TEMPLATE}/scripts/dock_submit.sh` in your own docking directory. You will need to change the path in `1_submit.sh` to this path (e.g. you will need to change it to `/path_to_docking_dir/dock_submit.sh` . Note that you will need to change this path in `1_submit.sh` every time you use wizard to do docking

- In your **own version** of `dock_submit.sh`, you should edit the file to look something like below: green are lines that need to be added, red are lines that need to be changed from the original.

```
#!/bin/bash

# this is just $DOCKBASE/docking/submit/submit.csh
# but saves the SGE job id to submit.pid


#$DOCKBASE/docking/submit/subdock.csh $DOCKBASE/docking/DOCK/bin/dock.csh

#if [ ! -f "dirlist" ]; then
#    echo "Error: Cannot find dirlist, the list of subdirectories!"
#    echo "Exiting!"
#    exit 1
#fi

if [[ ${CLUSTER_TYPE} = "SGE" ]]; then
    DIRNUM=$(wc -l dirlist)
    SUBMIT_JOB_ID=$(qsub \
      -terse \
      -t 1-$DIRNUM \
      $DOCKBASE/docking/submit/rundock.csh \
      $DOCKBASE/docking/DOCK/dock.csh)
    echo "Your job-array ${SUBMIT_JOB_ID}.1-${DIRNUM}:1 (\"rundock.csh\") has been submitted"
    echo "Saving SGE job id to submit.pid"
    echo $SUBMIT_JOB_ID > submit.pid
elif [[ ${CLUSTER_TYPE} -eq "SLURM" ]]; then
    # check if variables are defined
    # if the ${DOCKFILES} directory is writable, then create .shasum in in it
    # for each file in database.sdi
    #    if OUTDOCK.0 or test.mol2.gz.0 doesn't exist, add it to the joblist
    # call sbatch on rundock.bash

    export USE_DB2=true
    export USE_DB2_BATCH_SIZE=1
    export USE_DB2_TGZ=false
    export INPUT_SOURCE=$(readlink -f $1)
    export DOCKFILES=$(readlink -f $2)
    export EXPORT_DEST=$(readlink -f $3 )
    export USE_SLURM=true
    export DOCKEXEC=${DOCKBASE}/docking/DOCK/dock64
    export SHRTCACHE=${SCRATCH_DIR}
    export LONGCACHE=${SCRATCH_DIR}
    export SBATCH_ARGS="--account=${SLURM_ACCOUNT} --mail-user=${SLURM_MAIL_USER}
--mail-type=${SLURM_MAIL_TYPE} --partition=${SLURM_PARTITION}"

    DOCKFILES_COMMON=${SCRATCH_DIR}/DOCK_common/dockfiles.$(cat ${DOCKFILES}/* | sha1sum | awk
'{print $1}')
    echo "Copying dockfiles to '${DOCKFILES_COMMON}'"
    cp -r ${DOCKFILES} ${DOCKFILES_COMMON}
    #    njobs=$(wc -l dirlist)

    #    sbatch ${SBATCH_ARGS} --signal=B:USR1@120 --array=1-${njobs}
${DOCKBASE}/docking/submit/slurm/rundock.bash
    bash ${DOCKBASE}/docking/submit/subdock.bash
    echo "Check status with: squeue | grep -e \"$(whoami)\" -e \"rundock\""
else
    echo "Unrecognized cluster type '${CLUSTER_TYPE}'"
fi
```

- Make sure that the docking ran correctly by `cd` into the `results` directory and viewing one of the `test.mol2.gz.0` files. The file should be a binary file but should not contain just one line. If something ran incorrectly, you can check the `log` files to determine what the error was

Changing dock parameters:

- Editing the dock submit parameters:
    - To control the batch size of your docking runs (for example if you are docking multiple receptors at once and don't want to exceed job limit) you can edit the `dock_submit.sh` file to increase or decrease the batchsize as necessary **(make sure to make a copy of this file in your own directory and only edit that one)**

Analyzing the docking results:

- Once docking is complete, you need to analyze the results by:
- `cd` into the directory where the docking was run (the one that contains the `results` directory)
- Use `ls -d results/* > dirlist` to make a file called `dirlist`
- Remove all the items from the `dirlist` except for the directories containing the `test.mol2.gz.0` files … i write a shell script for cleaning the `dirlist` files, something like:

```
#!/bin/bash
sed -i '/file_list/d' dirlist
sed -i '/joblist/d' dirlist
sed -i '/logs/d' dirlist
```

- Once the `dirlist` is created and cleaned, run

```
python $DOCKBASE/analysis/extract_all_blazing_fast.py dirlist extract_all.txt 100
```

    - The highlighted stuff is what you will need to give as arguments (`dirlist`, name of file to be written (`extract_all.txt`), and max energy to accept (should match the energy cutoff in `INDOCK`)
    - This will output three files containing the energy scores for the docked poses in your results directory. The one that you probably want is the `extract_all.sort.uniq.txt` file. This is a file that contains the unique docked poses sorted by total energy (lowest energy → highest scoring poses)

- After running the extract python script, you should then run the script to get the mol2 files of the docked poses:

```
python $DOCKBASE/analysis/getposes_blazing_faster_py3.py '' extract_all.sort.uniq.txt 1000000 poses.mol2 test.mol2.gz.0
```

    - The highlighted things are the arguments you need to provide: the directory where docking is located, type ‘’ if you are in the docking directory, the file you want to get poses from (usually the `extract_all.sort.uniq.txt` file), the number of poses to get (set to a large number if you want to get all the poses), the name of the mol2 file to write, and the name of the mol2 files in the results directory (usually `test.mol2.gz.0`)
    - This script outputs a `poses.mol2` file that contains all the docked poses. You can then view these poses docked to the receptor in pymol or chimera by opening the `rec.pdb` and `poses.mol2` file together
    - For choosing docking hits, I usually choose the lowest 1% total energy poses (lower energy = better scoring) as ligands that are docked and the remaining 99% I classify as non-docking

Helpful scripts:

- I made a directory in `turbo` called `turbo/scripts_forDOCKing` that has some helpful scripts for docking many receptors and ligands. You can also look in `turbo/MPProjects/chemical_space/dock_dev/` for more scripts for batch docking and example docking runs