CLASH1A_28673486

*tar -xzvf namefile.tar.gz
namefile = your file
(In the following some parameters for CLASH1A_28673486 are listed)

* listobs: to identify targets and amplitude/phase calibrators
3C 48 --> amplitude/bandpass calibrator
J0132-1634 --> phase
A209 --> target
J0241-0815 --> phase
A383 --> target

* plotants: to select reference antenna --> ea05

* plotms averaging in time (10000), field=target, colorized per spw, iterated per antenna: to inspect the raw data and see the presence of RFI

-----A priori calibration and flagging----

* Antenna position correction
gencal(vis='namefile.ms', caltable='namefile.pos', caltype='antpos')

* Gain curve and opacity correction
The effects are very small (less than or about 1%) across the frequency range of this observation (1-2 GHz) --> no opacity correction is done

*Flagging non-operational antennas

flagdata(vis='namefile.ms/', mode='manual', antenna='ea01, ea25')

* Flagging shadowed antennas and zero-amplitude data

There may be instances where one antenna blocks, or "shadows" another.
flagdata(vis='namefile.ms/', mode='shadow', flagbackup=False)

There may be times during which the correlator writes out pure zero-valued data. In order to remove this bad data, we run flagdata to remove any pure zeroes:
flagdata(vis='namefile.ms/', mode='clip', clipzeros=True, flagbackup=False)

* Hanning-smoothing data
This function Hanning smoothes the frequency channels with a weighted running average. The weights are 0.5 for the central channel and 0.25 for each of the two adjacent channels. The first and last channels are flagged.

Plotms on the target, and selecting some spectral windows. Save the plot in amp_v_freq.beforeHanning.png

```
plotms(vis='namefile.ms', field ='2', antenna='ea05', spw='0~4', xaxis='freq', yaxis='amp',
coloraxis='spw',correlation='RR,LL', plotfile='amp_v_freq.beforeHanning.png')
```

```
hanningsmooth(vis='namefile.ms/', outputvis='namefile_smooth.ms',datacolumn ='data')
```

It removed the effects of some of the worst RFI from a number of channels. Overall, this will improve our ability to flag RFI from the data and retain as much good data as possible.

check the difference in data before and after the smoothing. Put the plot axis in the same range of the amp_v_freq.beforeHanning.png and save the image in amp_v_freq.afterHanning.png

```
plotms(vis='namefile_smooth.ms', field ='2', antenna='ea05', spw='0~4', xaxis='freq', yaxis='amp',
coloraxis='spw',correlation='RR,LL', plotfile='amp_v_freq.afterHanning.png')
```

* Using the phase calibration source for preliminary bandpass calibration
Use one of the 2 phase calibrators you have: it is not necessary do this calibration on all phase calibrators

Since the RFI is time-variable, using the phase calibration source to make an average bandpass over the entire observation will mitigate the amount of RFI present in the calculated bandpass.
Choose a narrow range of channels for each spectral window which are relatively RFI-free

Plotms(ch,amp) on the target iterated per spw to find the bad channels with RFI

```
plotms(vis='namefile_smooth.ms', field ='2', antenna='ea05', xaxis='channel', yaxis='amp',
iteraxis='spw', correlation='RR,LL')
```

In the following, the bad channels for each spw

```
spw0 19~25 31~36
spw1 15~40 45~60
spw2  4 52~57
spw3 2~10 19~36 42~47
spw4 ok
spw5 ok
spw6 ok
spw7 ok
spw8 flagged
spw9 15~21 33~46
spw10 25~50
spw11 ok
spw12 ok
spw13 ok
spw14 25~47 58~60
spw15 11~19
```

Therefore, a set of narrow (6ch) range spw without RFI could be

```
spw='0:40~45, 1:5~10, 2:5~10,
```

3:12~17,4:30~35,5:30~35,6:30~35,7:30~35,9:25~30,10:10~15,11:30~35,12:30~35,13:30~35,14:49~54,15:30~35'

gaincal(vis='namefile_smooth.ms', caltable='namefile_smooth.initPh', field='1', spw='0:40~45, 1:5~10, 2:5~10,3:12~17,4:30~35,5:30~35,6:30~35,7:30~35,9:25~30,10:10~15,11:30~35,12:30~35,13:30~35,14:49~54,15:30~35', solint='int', refant='ea05', minblperant=3,minsnr=3, calmode='p',gaintable='namefile.pos')

Notes on gaincal:
solint='int': we request a solution for each 10-second integration.
minblperant=3: the minimum number of baselines which must be present to attempt a phase solution.
minsnr=3.0: the minimum signal-to-noise a solution must have to be considered acceptable. Note that solutions which fail this test will cause these data to be flagged downstream of this calibration step.
calmode='p': perform phase-only solutions.
gaintable='namefile_smoooth.pos': use the antenna position correction that we created earlier.

Plot time vs phase iterating over antenna for a single spectral window. We can see that the phase does not change much over the course of the observation.
plotcal(caltable='namefile_smooth.initPh', xaxis='time',yaxis='phase', iteration='antenna',spw='6',plotrange=[-1,-1,-180,180])

Using this phase information, we create time-averaged bandpass solutions for the phase calibration source:
bandpass(vis='namefile_smooth.ms', caltable='namefile_smooth.initBP', field='1', solint='inf', refant='ea05', minblperant=3,minsnr=10, gaintable=['namefile.pos','namefile_smooth.initPh'], combine='scan', interp=['','nearest'],solnorm=False)

Notes on bandpass:
solint='inf', combine='scan': the solution interval of 'inf' will automatically break by scans; this requests that the solution intervals be combined over scans, so that we will get one solution per antenna.
gaintable=['namefile.pos', 'namefile_smooth.initPh']: we will pre-apply both the antenna position corrections as well as the initial phase solutions.
interp=['', 'nearest']: by default, gaincal will use linear interpolation for pre-applied calibration. However, we want the nearest phase solution to be used for a given time.

We may plot the bandpas solutions with plotcal; first looking at the amplitudes:
plotcal(caltable='namefile_smooth.initBP', xaxis='freq',yaxis='amp', iteration='antenna',subplot=331)

Notes on plotcal:
subplot=331: displays 3x3 plots per screen

Also, we can look at the phase solutions:
plotcal(caltable='namefile_smooth.initBP', xaxis='freq', yaxis='phase', iteration='antenna', subplot=331)

Check if some spw are wiped-out by RFI or some channels are consistently badly affected. If yes, flag manually
e.g. flagdata(vis='namefile_smooth.ms', spw='bad ch')

Now we apply the antenna position corrections and the bandpass calibration table to the data. This operation will flag data that correspond to flagged solutions, so applycal makes a backup version of the flags prior to operating on the data
applycal(vis='namefile_smooth.ms', gaintable=['namefile.pos', 'namefile_smooth.initBP'], calwt=False)

To see the corrected data, we can plot the data of the target as we did before, choosing ydatacolumn='corrected' this time:
plotms(vis='namefile_smooth.ms', field='2', antenna='ea05', xaxis='freq', yaxis='amp', coloraxis='spw', iteraxis='scan', ydatacolumn='corrected')

* Automatic flagging

To save original data and come back to them if the further flags are not good.

```
flagmanager(vis = 'namefile_smooth.ms',
          mode = 'save',
             versionname = 'Original')
```

```
# In CASA
default flagdata
mode='rflag'
```

rflag = automatic detection of outliers based on sliding-window RMS filters extend

```
flagdata(vis='namefile_smooth.ms', mode='rflag', field='2',
        spw='0', datacolumn='corrected',
        freqdevscale=5, timedevscale=5,
        action='calculate', display='both',
        flagbackup=False)
```

Let's try making it more sensitive to deviations from the calculated RMS in frequency, setting both timedevscale and freqdevscale=1.5 (the default is 5.0):

```
flagmanager(vis = 'namefile_smooth.ms',
          mode = 'save',
             versionname = 'automaticRFIflag')
```

* Manual flagging of the RFI

Do the manual flag with flagdata and then save the manual flags with flagmanager

```
flagmanager(vis = 'namefile_smooth.ms',
          mode = 'save',
             versionname = 'manualRFIflag')
```

* Evaluating results & further manual flagging

One instructive ways to view the data of the target are by baseline and uv-distance. Note that we're plotting all baselines in these plots, rather than just baselines to ref antenna as before.

```
# In CASA
```

```
plotms(vis='namefile_smooth.ms', field = '2',
       xaxis='baseline', yaxis='amp', iteraxis='spw',
       correlation='RR,LL', coloraxis='antenna1')
```

If no particular baselines look bad enough to flag outright, so we will leave this as is. Feel free to do some more flagging if you like. Now, let's plot as a function of uv-distance:

```
# In CASA
plotms(vis='namefile_smooth.ms', field ='2',
       xaxis='uvdist', yaxis='amp', iteraxis='spw',
       correlation='RR,LL', coloraxis='antenna1')
```

Again, feel free to do some more flagging if you think it is necessary.

** Calibrating data
 * Setting the flux density scale

Since we will be using 3C48 as the source of the absolute flux scale for this observation, we must first run setjy to set the appropriate model amplitudes for this source.
If the flux calibrator is spatially resolved, it is necessary to include a model image as well. In any case, we suggest to include the model image for completeness.

First, we use the listmodimages parameter to find the model image path:

```
setjy(vis='namefile_smooth.ms', listmodels=True)

setjy(vis='namefile_smooth.ms', field='0', scalebychan=True, spw='', model='3C48_L.im')
```

Notes on setjy
scalebychan=True: scales the model flux density value for each channel. By default, only one value per spectral window is calculated.

* Delay and Bandpass calibration
We will follow a similar procedure as the one outlined above for calculating the antenna bandpasses, except that this time, we will use the actual designated bandpass calibration source, 3C48. Although the phase calibration source has the advantage of having been observed throughout the run, it has an unknown spectrum which could introduce amplitude slopes to each spectral window. We will also calibrate the residual antenna-based delays

As before, we first generate a phase-only gain calibration table that will be used to help smooth-out the phases before running bandpass itself:
select spw and channel of the bandpass calibrator free of RFI --> if spw is all good, keep it entirely

```
gaincal(vis='namefile_smooth.ms', field ='0',
        caltable='namefile+smooth.initPh.2',
        spw='good channel',
        solint='int', refant='ea05',
        minblperant=3, minsnr=3.0, calmode='p',
        gaintable='namefile.pos')
```

If you will notice a lot of messages that read "Insufficient unflagged antennas to proceed with this solve" for SPW n. This indicates that too much data have been flagged to perform the gaincal operation. This also suggests that the spectral window is too badly affected by RFI to be useful for imaging -- so, flag the rest of the SPW before continuing with further analysis:
e.g.

```
flagdata(vis='namefile_smooth.ms', spw='n')
```

We can now solve for the residual antenna-based delays that can be seen in plots of the phase vs. frequency for the calibrator sources in plotms. This uses the gaintype='K' option in gaincal. Note that this currently does not do a "global fringe-fitting" solution for delays, but instead does a baseline-based delay solution to all baselines to the refant, treating these as antenna-based delays. In most cases with high-enough S/N to get baseline-based delay solutions this will suffice. We use our bright bandpass calibrator, 3C48, to calibrate the delays:

```
# In CASA
gaincal(vis='namefile_smooth.ms', field ='0',
        caltable='namefile_smooth.K0',
        spw='good chan', solint='inf', refant='ea05',
        gaintype='K', combine='scan', minsnr=3,
        gaintable=['namefile.pos','namefile_smooth.initPh.2'])
```

We pre-apply our initial phase table, and produce a new K-type caltable for input to bandpass calibration. We can plot the delays, in nanoseconds, as a function of antenna index (you will get one for each sub-band and polarization):

```
plotcal(caltable='namefile_smooth.K0', xaxis='antenna', yaxis='delay')
```

Now let's solve for the bandpass using the previous tables:

```
# In CASA
bandpass(vis='namefile_smooth.ms', caltable='namefile_smooth.bPass',
         field='0', solint='inf', spw='',
         combine='scan', refant='ea05', minblperant=3, minsnr=10.0,
         gaintable=['namefile.pos','namefile_smooth.initPh.2','namefile_smooth.K0'],
         interp=['', 'nearest', 'nearest'], solnorm=False)
```

Notes on bandpass:
solint='inf', combine='scan': again, the solution interval of 'inf' will automatically break-up the data by scans; this requests that the solution intervals be combined over scans, so that we will get one solution per antenna. Note that you must set solnorm=False here or later on you will find some offsets between spws due to the way in which amplitude scaling adjusts weights internally during solving.

Note that since we have flagged-out the vast majority of the RFI-affected data, there may are many fewer failed solutions. Again, we can plot the calculated bandpasses to check that they look reasonable:

```
plotcal(caltable='namefile_smooth.bPass', xaxis='freq', yaxis='amp',
        iteration='antenna', subplot=331)
plotcal(caltable='namefile_smooth.bPass', xaxis='freq', yaxis='phase',
        iteration='antenna', subplot=331)
```

* In the next steps will be split: first we calibrate phase cal 1 and target 2, then phase cal 3 and target 4

Next, we will calculate the per-antenna gain solutions. Since this is low-frequency data, we do not expect substantial variations over short timescales, so we calculate one solution per scan (using "solint='inf'"):

```
gaincal(vis='namefile_smooth.ms', caltable='namefile_smooth.phaseAmp', field ='0,1,3',
        spw='', solint='inf', refant='ea05', minblperant=3,
        minsnr=10.0, gaintable=['namefile.pos','namefile_smooth.K0','namefile_smooth.bPass'])
```

Plot these solutions as a function of amplitude and phase versus time for the phase calibrator (field 1 and 3), iterating over each antenna:

```
# In CASA
plotcal(caltable='namefile_smooth.phaseAmp', xaxis='time', yaxis='amp',
        field = '1', iteration='antenna')
plotcal(caltable='namefile_smooth.phaseAmp', xaxis='time', yaxis='phase',
        field = '1', iteration='antenna')
```

* Flux scaling the gain solutions

Now that we have a complete set of gain solutions, we must scale the phase calibrator's absolute flux correctly, using 3C48 as our reference source. To do this, we run fluxscale on the gain table we just created, which will write a new, flux-corrected gain table:
Fluxscale is used bootstrap the flux density scale from standard calibrators.
After running gaincal on standard flux density calibrators (with or without an image model), and other calibrators with unknown flux densities (assumed 1 Jy), fluxscale applies the constraint that net system gain was, in fact, independent of field, on average, and that field-dependent gains in the input caltable are solely a result of the unknown flux densities for the calibrators. Using time-averaged gain amplitudes, the ratio between each ordinary calibrator and the flux density calibrator(s) is formed for each antenna and polarization (that they have in common). The average of this ratio over antennas and polarizations yields a correction factor that is applied to the ordinary calibrators' gains.

```
fluxscale(vis='namefile_smooth.ms', caltable='namefile_smooth.phaseAmp',
          fluxtable='namefile_smooth.phaseAmp.f.Scale', reference='0', incremental=False)
```

Note that the myFlux Python dictionary will contain information about the scaled fluxes and fitted spectrum. The logger will display information about the flux density it has deduced for phase cal:
Check that the values listed are similar to those in the VLA Calibrator Manual.

* Applying calibration

Finally, we must apply the calibration to our data. To do this, we run applycal in two stages: the first is to self-calibrate our calibration sources; the second, to apply calibration to the supernova remnant. These must be done separately, since we want to use "nearest" interpolation for the self-calibration and "linear" for the application to the science target:

For target do applycal specifying for the .fScale table the gainfield = relative phase calibrator

```
applycal(vis='namefile_smooth.ms', spw='', field = '2',
              gaintable=['namefile.pos','namefile_smooth.K0','namefile_smooth.bPass',
'namefile_smooth.phaseAmp.fScale'], calwt=False,
              gainfield = ['','','', '1'])

applycal(vis='namefile_smooth.ms', spw='', field = '4',

gaintable=['namefile.pos','namefile_smooth.K0','namefile_smooth.bPass','namefile_smooth.phaseAmp.
fScale'], calwt=False,
               gainfield = ['','','', '3'])

applycal(vis='namefile_smooth.ms', spw='', field='0,1,3',
         gaintable=['namefile.pos','namefile_smooth.K0','namefile_smooth.bPass',
                    'namefile_smooth.phaseAmp.fScale'],
         calwt=False, interp=['','nearest','nearest','nearest'])
```

* Plotting calibrated data
e.g.

```
plotms(vis='namefile_smooth.ms', field='1', xaxis='imag', yaxis='real',
       xdatacolumn='corrected', ydatacolumn='corrected', coloraxis='antenna1',
       avgchannel='10', avgtime='20', correlation='RR,LL', iteraxis='spw',
       spw='')

plotms(vis='namefile_smooth.ms', field='1', xaxis='baseline', yaxis='amp',
       xdatacolumn='corrected', ydatacolumn='corrected', coloraxis='antenna1',
       avgchannel='10', avgtime='20', correlation='RR,LL', iteraxis='spw',
       spw='')
```

* Splitting out data for corrected data

```
split(vis='namefile_smooth.ms', field='', keepflags=False,
      outputvis='calib.ms', datacolumn='corrected',
      spw='', correlation = 'RR,LL')
```

*Imaging

select the field you want image.

```
clean(vis='calib.ms', imagename ='field_n', imsize=256, field ='n'
      cell = '0.3 arcsec', mode ='mfs', niter = 1000,
      interactive = True)
```

viewer: to see final cleaned images.