GEOS-Chem Model Clinic

05 May 2011

Bob Yantosca
GEOS-Chem Support Team
geos-chem-support@as.harvard.edu

Table of Contents

Git Version Control System

Why version control?

- G-C is a complex model
- Many users are submitting updates for inclusion into G-C
- Several lines of development are open simultaneously
- Need to trace the revision history of the code

G-C now uses Git for version control

- We migrated from CVS to Git in 2010
- We now use Git to archive G-C source code, run dirs, and GAMAP
- See G-C manual Ch. 2

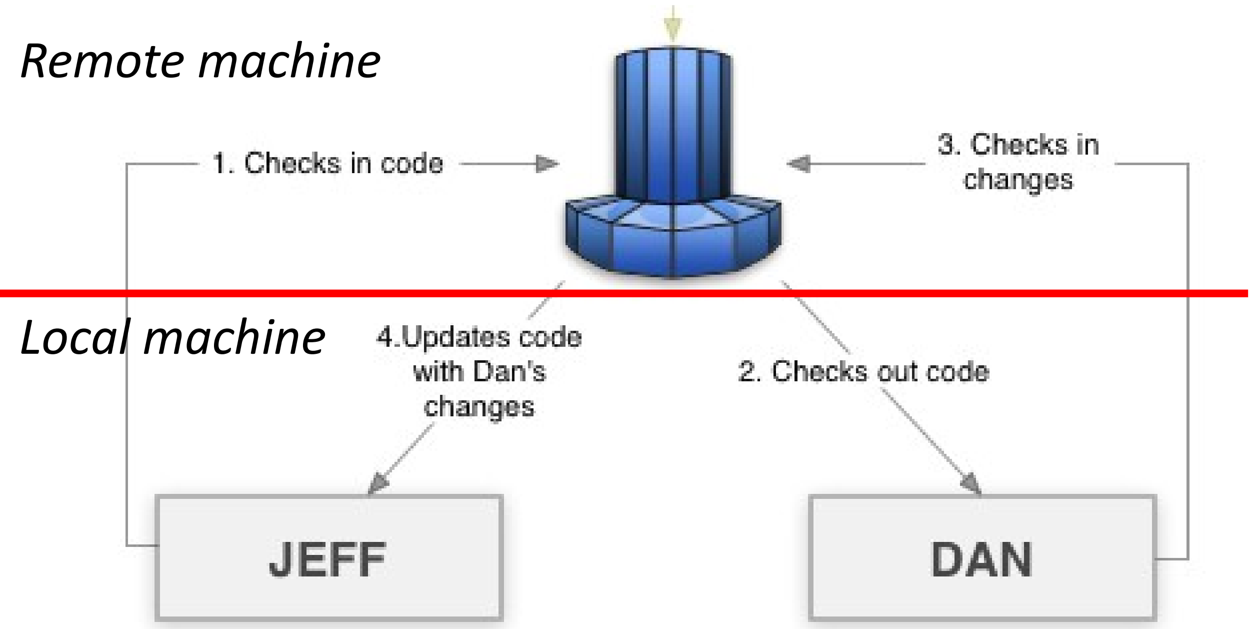
Overview of Git

Git ...

- was created by Linus Torvalds to do the version control for Linux
- is a **distributed** source code mgmt system
- allows you to easily combine code from several developers
- makes branching and merging trivial
- contains easy-to-use graphical tools
- does not have the problems of older version-control software (e.g. RCS, CVS, Subversion)

Before Git, there was CVS...

Centralized repository
(e.g. RCS, CVS, Subversion)



Reference: http://hoth.entp.com/output/git_for_designers.html

This schematic represents the older generation of version control software.

Each user checks out the "standard" source code files from a single master repository. Then each user modifies his/her code locally and uploads (or "pushes") their changes back to the central repository.

Problems:

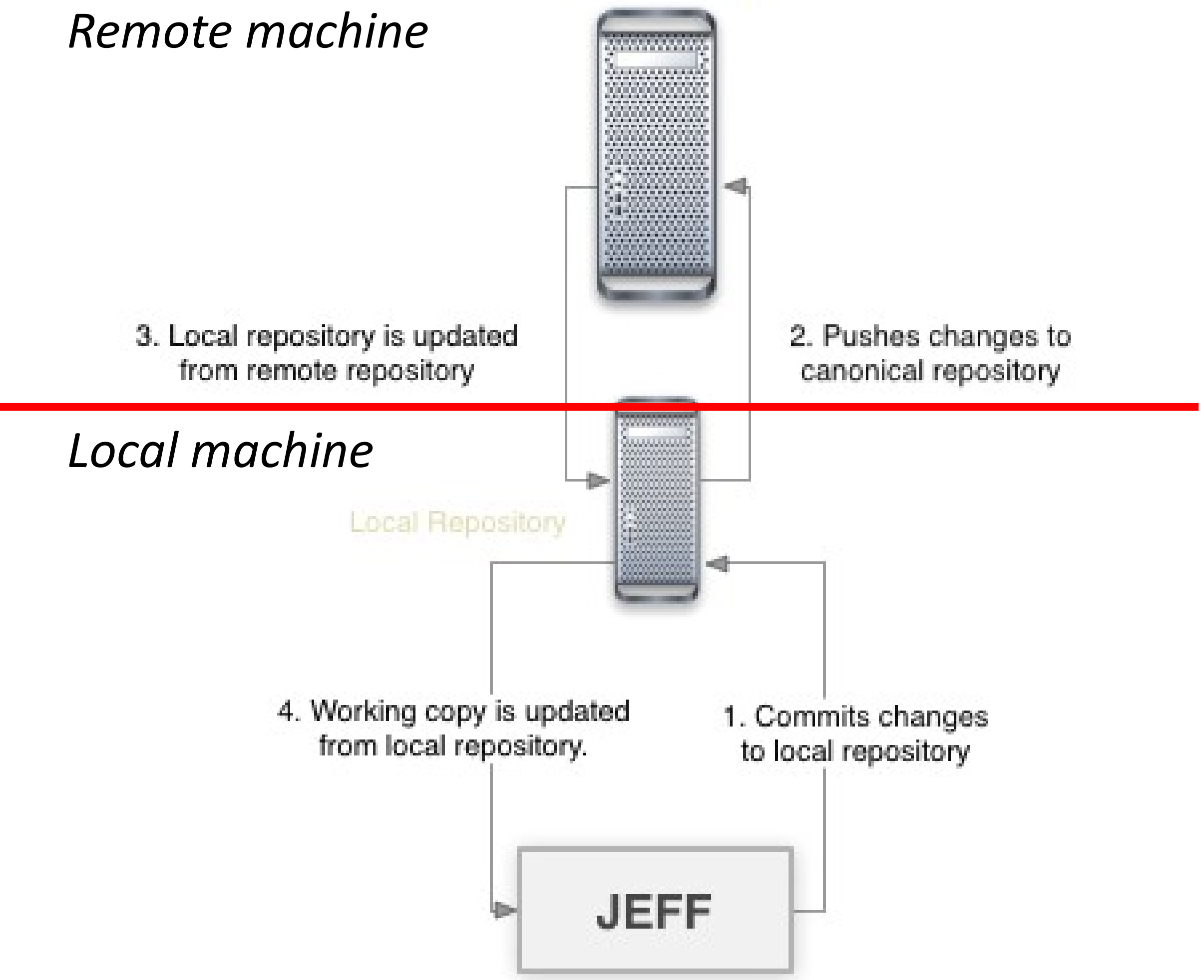
- Single point of failure
- Potential permission problems
- Naming confusion
- No local access to revision history

Git is a decentralized system

Decentralized repository (e.g Git)

Remote machine

Local machine


3. Local repository is updated from remote repository
2. Pushes changes to canonical repository
Local Repository
4. Working copy is updated from local repository.
1. Commits changes to local repository
JEFF

Reference: http://hoth.entp.com/output/git_for_designers.html

This schematic represents a decentralized version control system such as Git.

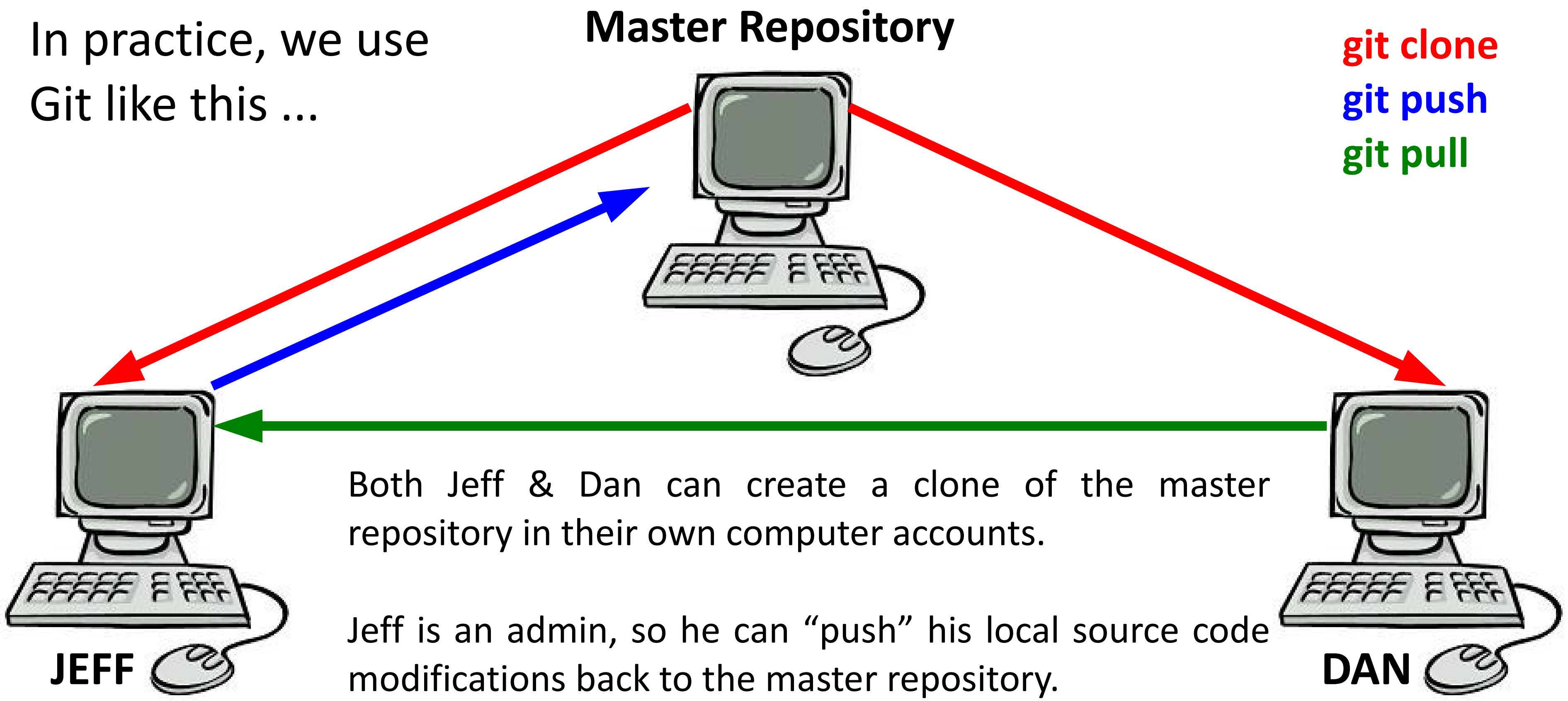
Each user makes a “clone” of the remote repository on his/her local system. This clone not only contains the source code files, but also the total revision history going back to the start of the project.

Each user then modifies his/her local copy of the repository. Changes made locally do not affect the “master” repository.

Users w/ the proper permission can “push” changes back to the “master” repository.

Practical use of Git

In practice, we use Git like this ...



Both Jeff & Dan can create a clone of the master repository in their own computer accounts.

Jeff is an admin, so he can "push" his local source code modifications back to the master repository.

Dan does not have admin privileges, so he sends his modifications to Jeff (via git pull or patch file). Jeff merges Dan's changes into his local repository, and then "pushes" them back to the master repository.

Brief recap

With Git …

- each clone is a backup of the “master” repository
- work can continue if an asteroid wipes out the East Coast!
- no repository is inherently more important than any other
- in practice, one repository is denoted to be the “master”, from which all users agree to download code updates.
- each user can see the full revision history of the code, starting from day one!
- Modifications made to a “cloned” repository will not affect the “master” repository.
- In practice, only admins have special permission to update the “master” repository.

Downloading & updating

git clone

- Gives you a clean copy of the “master” repository
- You get source code files + total version history
- Use this to download a new version of the code

git pull

- Gets updates from the “master” repository and puts them into your existing local repository
- Any work you have already started in your local repository will not be discarded.
- Use this to apply bug fixes (aka “patches”) to your code, or to get code from another user's local repository (e.g. from Dan to Jeff)

NOTE: We recommend that you do not pull updates into the master branch. You can create a new branch into which to receive the updates (more on that later).

Updating the master repository

git push

- Uploads code from your local repository to the “master” repository
- Typically this will only be used by people w/ admin privileges

Let's try Git with the GEOS-Chem code!

Git's graphical tools

gitk

- Lets you browse the complete revision history
- Shows all of the commits that were made to the repository
- For each commit, you can see the changes made to each file
- You can see the branching history of the repository

git gui

- Graphical user interface for most Git commands:
- Creating new branches of development
- Committing (and documenting!) changes to the repository
- Merging branches back into the “master” line of development

We recommend using the git gui rather than typing in the commands at the Unix prompt. The git gui greatly simplifies things.

What Git uses to track history

Git uses 4 types of objects:

- ***Blobs*** (i.e. source code files and directories)
- ***Commits*** (the state of the entire code at a given time)
- ***Merges*** (the joining of 2 commits on separate branches)
- ***Tags*** (an alpha-numeric phrase attached to a commit or merge)

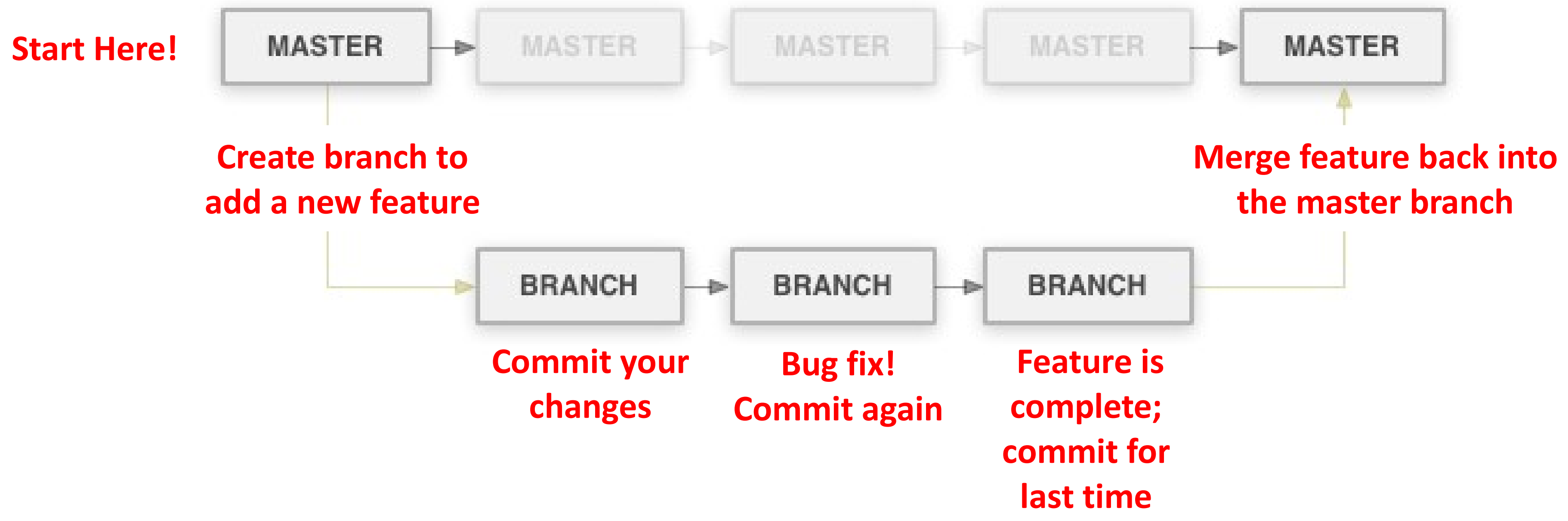
In the gitk upper left-window:

- Commits are blue dots connected by lines
- Colored lines represent the various branches of development
- Branch names are shown in green boxes
- Lines show the previous (parent) & successive (child) commits
- A merge is a special type of “commit” where 2 branches join
- Tags are denoted by text in a yellow box

Branching with Git

Git allows you to split model development from the main line of development (aka "master branch") into one or more branches.

Each branch can contain a new feature of the code. Branches also allow features to be tested independently of each other. When a feature is complete, its branch can be merged back into the "master branch". After a merge, the "master branch" will contain the previous code (i.e as of when the branch was first made), plus the new feature.


Start Here!
MASTER
MASTER
MASTER
MASTER
MASTER
Create branch to add a new feature
Merge feature back into the master branch
BRANCH
BRANCH
BRANCH
Commit your changes
Bug fix! Commit again
Feature is complete; commit for last time

Reference: http://hoth.entp.com/output/git_for_designers.html

Git's branching commands #1

Creating a branch

- In git gui: ***Branch / Create***
- In git gui, the branch is also checked out after it is created

Checking out a branch

- In git gui: ***Branch / Checkout***
- Allows you to switch between branches.
- The checkout command switches the files in your source code directory so that they correspond to the branch you requested.
- Uncommitted files will not be changed.
- To avoid confusion, you should always commit your updates to the current branch before checking out a new branch.

NOTE: After you create/checkout your branch, you will want to start working there and commit your changes to that branch...

Commiting changes w/ Git

Committing is best done w/ the Git Gui

- ***Unstaged changes*** (top-left GUI window): files w/ updates that Git does not know about yet. Click on them to stage.
- ***Staged changes*** (bottom-left GUI window): files that will be added to the local repository when you commit.
- ***Commit message*** (lower-right GUI window): You can type a message describing the files/directories that are being committed. The first line of the message will show up as the commit title in the GitK browser.
- ***Sign-off button*** (lower-right GUI window): Click this button to add your name & email to the commit message.
- ***Commit button*** (lower-right GUI window): Click this button to commit your updated files/directories to the repository.
- ***Amend Last Commit*** (lower-right GUI window): Check this box if you want to update the message from the last commit.

Git's branching commands #2

Merging branches together

- In git gui: ***Merge / Local Merge***
- Then pick the name of the branch that you want to merge into the current branch.

Deleting branches

- In git gui: ***Branch/Delete***
- Once you have merged a branch, you can delete it.

NOTE: Hitting F5 will cause both gitk and git gui to refresh. You should do this often, so that the gitk and git gui will “see” all of the recent changes made to the repository.

Sharing code with others

Git pull

- When you can “see” the other repository on disk or via internet
- Recipient “pulls” code from sender's repository

Send a patch file via email

- Patch file = file containing the “diffs”, or changes between commits
- Create with ***git format-patch*** from the command line
- Attach to email and send to person you want to share code with
- Recipient uses ***git am*** command to ingest changes into his/her local repository

NOTE: For a patch file to work, you have to know its parent commit.

Sharing code with others

The sender of the patch file will type this at the Unix prompt:

```
git format-patch -2 --stdout > my_patch_file.txt
```

Where the -2 is the # of commits (starting w/ the most recent) to include in the patch file (set this to how many you want). The text file can then be sent to the recipient by email.

The recipient of the patch will type this at the Unix prompt:

```
git am < my_patch_file.txt
```

This will add the commits contained in the patch file to the recipient's local repository. NOTE: the parent commit in the recipient's repository has to be the same as the parent of the last commit contained w/in the patch file.

Compiling GEOS-Chem

Some important compiling commands:

- ***make help*** : shows help screen with all possible makefile options
- ***make -j4:*** compiles 4 files simultaneously (you can pick a different #)
- ***make clean:*** Removes *.o *.mod files in source code subdirs only
- ***make realclean:*** Removes all *.o *.mod *.a *.lib etc. files everywhere
- ***make -j4 DEBUG=yes TRACEBACK=yes:*** Compile for use w/ debugger
- ***make -j4 BOUNDS=yes:*** Compile w/ array-out-of-bounds checking
- ***make doc:*** Builds the G-C reference docs with the ProTeX script
- ***make -j4 lib:*** Only compiles the code but does not build executable
- ***make -j4 exe:*** Builds the executable

NOTE: Executable file is produced in both the ***GeosCore*** and ***bin*** subdirectories of the source code directory.

See Ch. 3 of the GEOS-Chem manual for detailed info

GEOS-Chem run directories

You can download a G-C run directory with a git clone operation:

```
git clone git://git.as.harvard.edu/bmy/GEOS-Chem-rundirs/DIRNAME
```

Where DIRNAME is a combination such as:

```
4x5/geos5/standard
4x5/geos5/SOA
4x5/geos5/dicarbonyls, etc.
```

The most important input files in the run directories are ***input.geos*** (sets run options), ***globchem.dat***, and ***mglob.dat*** (chemical mechanism).

You also need to copy the executable file from the source code directory to the run directory.

See Ch. 2 and Ch. 5 of the GEOS-Chem manual for detailed info.

Setting run options

Check the following settings in your input.geos file:

- Start and end dates of run
- Directories where data files are located
- Restart file names
- Transport options
- Convection options
- Emissions options
- Deposition options
- Chemistry options
- Diagnostic options
- Timeseries diagnostic options
- Nested-grid options (if necessary)

Debugging GEOS-Chem

If you suspect something is wrong w/ your run:

- Try recompiling with ***BOUNDS=yes TRACEBACK=yes***
- ***BOUNDS=yes*** : turns on out-of-bounds checking
- ***TRACEBACK=yes*** : will return the error stack (list of all routines that were called when the error happened)

Use a debugger if you have one!

- Most users: idb or dbx
- Lucky users: totalview
- Compile with ***DEBUG=yes TRACEBACK=yes OMP=no***
- ***DEBUG=yes*** : turns off optimization (necessary for debugging)
- ***OMP=no*** : turns off parallel processing

Visualizing G-C output

Use GAMAP for most G-C output visualization tasks

- See http://acmg.seas.harvard.edu/gamap for more info

GAMAP subroutines can be used as standalones

- CTM_GET_DATA (gets several data blocks)
- CTM_GET_DATABLOCK (gets 1 data block; cuts it to size)
- TVMAP (plots data on a lat-lon world map)
- TVPLOT (plots lat-alt or lon-alt data)
- CTM_PLOT (reads data from disk and plots it)
- CTM_SUM_EMISSIONS (sums up emissions totals)
- etc

Q & A

- Topics that you want to discuss?
- Demos that you'd like to see?