# Instructions on Using Git

## 1. Basics

Git is the current most popular version control system. It is similar to CVS in many ways, but there are some differences. Most importantly each checked out git repository has full version history by default and it allows local commits. Overall Git has more features and more flexibility than CVS. We convert the current CVS repositories to Git in order to provide a more effective and up-to-date version control environment to our users and developers. An effort is made to preserve the most important functionalities of the current CVS repository, including the ability to check out the whole SWMF as well as the stand-alone models.

## 2. Structure of the SWMF Git Repository

The version control information is stored at the top level of the working directory which is different from the CVS system that stores information in each directory. To accommodate the SWMF structure where models can be checked out in stand-alone mode as well as part of the SWMF, we use multiple Git repositories and the Git *submodules*. We created several Git repositories (SWMF.git, BATSRUS.git, GITM2.git, share.git, util.git and so on) and use Git submodules to include them into a whole SWMF Git repository. A Git submodule is no more than just a separate Git repository with version and address information stored in its parent repository (in this case, the SWMF). All the SWMF related Git repositories are stored in the

herot.engin.umich.edu:/GIT/FRAMEWORK/

directory.

## 3. Set up with the ssh connection to herot.engin.umich.edu

Make sure that you do not need to type your password to access herot unless you want to type it hundreds of times. If you haven’t done this for CVS, now you really should. Use

$ ssh-keygen # just hit return to all questions

$ scp ~/.ssh/id\_rsa.pub herot.engin.umich.edu: # use password to copy key to herot

$ ssh herot.engin.umich.edu # very last time you need password to login

$ cat id\_rsa.pub >> .ssh/authorized\_keys # append key to authorized keys

$ rm id\_rsa.pub # remove key from home directory

This should be done for all machines from which you want to access the Git repository.

## 4. Basic Commands



Figure The working logic of Git system

### Clone a remote git repository to a local one (similar to **cvs checkout**)

$ git clone SERVER\_ADDRESS:ADDRESS/REPONAME.git LOCALNAME

The optional LOCALNAME allows changing the name locally (not available in CVS).

The command above clones the remote repository to the local machine with complete version history. This is an improvement over CVS because the history can be accessed without connecting to the remote server. However, the size of the local repository will be about 900 MB for the SWMF instead of 187 MB size of the CVS repository. Using -depth=NUMBER allows limiting the version history, for example:

$ git clone --depth=1 --recursive herot:/GIT/FRAMEWORK/SWMF.git SWMF\_NO\_HISTORY

clones the SWMF with the latest version only and the size is reduced to about 400 MB. This reduces disk usage and initial download time.

### Check the difference between the present git repository and the remote server

When other developers modified the project and pushed this changes to the remote repository, to check the difference between the local repository and the remote, we recommend the workflow below:

$ git fetch origin master:temp

#download the remote branch ‘master’ to the local branch ‘temp’

$ git diff temp

# compare the local branch with temp (the newest version in remote)

$ git merge temp

#apply the changes to the local, conflicts may occur during the merging process, please refer to the internet for information about dealing with conflicts

$ git branch -d temp

# delete the branch ‘temp’, which is no longer useful

You may find another way (type less) to finish this process, but we recommend following this workflow, since you know what changes are applied to your local repository exactly. If you are confident with the remote repository

$ git pull

Will update your local repository (equivalent to fetch+merge)

### Make changes to local repository (similar but not the same as **cvs add/remove/commit**)

In essence use “git” followed by the Unix commands to remove and rename files and directories, use “add” to add a new file and “commit” to commit the changes into the local repository (note that CVS does not allow renaming files or removing directories and there is no local repository):

$ git rm -rf DIR1

$ git rm FILE1

$ git commit -m "removed FILE1 and DIR1 because …"

$ git mv FILE2 FILE3

$ git commit -m “renamed FILE2 to FILE3”

$ emacs FILE4

$ git add FILE4

$ git commit “Created FILE4 which does ….”

Without the -m flag the editor (defined by the $EDITOR environment variable) opens to allow logging the changes. Just like with CVS, meaningful logs that describe the changes, the reasons for them and the consequences are extremely important and useful. You can also commit files separately and provide separate logs. This is recommended if the changes in different files are not related to each other.

The steps above affect the local git repository only. This allows storing multiple versions with a complete version history locally without making changes in the remote repository.

### Check the status of the present git repository (similar to **cvs -n update -d**)

After working for a while, to check what you have done, type

$ git status

Also, you can use tkdiff to examine the changes relative to FILE1

$ tkdiff FILE1

### Push changes to remote repository

After a period of local development, you may want to push the changes to the remote server

Make sure you are on the master branch (same as **CVS HEAD**) in the local repository as well as in the submodules.

$ git push

### (5) Getting versions by date

**NOTICE: The function to get the history of entire SWMF and its submodules recursively is still under developing. Now a user is only allowed going back to an older version in a submodule.**

Sometimes is necessary to come back to an older version (for example for debugging). Notice that the commit history in each repository is separated from the others. To make things easy, we need to make an alias first (-- global means you only need to do the setup only once on this machine):

$ git config --global alias.hist "log --pretty=format:'%h %ad | %s%d [%an]' --graph --date=short"

If you want to see the commit on a specific day, saying 2014-01-01, grep is needed:

$ git hist | grep 2014-01-01

The commit histories will come out with a hash number. Type $ git checkout #hash will lead you to that version. For example:

$ git hist | grep 2014-01-01

\* 1f801a596 2014-01-01 | added optional compiling/linking of the models [vtenishe]

\* c7de35b78 2014-01-01 | Updated test result for corrected rotation period. [gtoth]

Now you can reset your code to the version 1f801a596:

$ git reset --hard 1f801a596

The command below allows you to get the old version of a single file in commit 1f801a596:

$ git checkout 1f801a596 path/to/your/file

Create a branch to check the older version is also an option, resources about create and merge the branches are available online.

## Working with the entire SWMF

Clone the entire SWMF repository (similar to **cvs checkout SWMF**) with all the submodules (note the --recursive flag):

git clone --recursive herot.engin.umich.edu:/GIT/FRAMEWORK/SWMF.git

If you forget to use the recursive flag, only the Empty versions will be present in each component directory and the share/ and util/ directories will be missing too. To get the missing submodules and their contents:

$ git submodule init

$ git submodule update --remote

Now that we have the submodules, we have to switch to the master branch (by default Git produces a local branch)! This step is required to be able to update from the remote repositories or to push changes into the remote repositories. Type

$ git submodule foreach 'git checkout master'

If someone worked on individual submodules not on SWMF, you may not get the newest version of submodule information in SWMF (while you can still get the newest version of each modules), now you need to add the changes in submodules and do a commit to record you update (in SWMF):

$ git add . && git commit -m “find submodule change by \*, make a commit in SWMF”

In addition, set the following option to get status information about the whole SWMF when typing git status:

$ git config --global status.submodulesummary 1

Then we can start working on this submodule, don't forget to commit the changes!

(5) Commit in SWMF

Because we organize the SWMF with submodules, we need to do the commit (let the SWMF know) after we made changes in submodules. Go back to the SWMF directory:

$ git add . #add all changes

$ git commit -m “modified SUBMODULE\_NAME”

(6) Pushing submodule changes (**MAKE SURE YOU ARE ON MASTER BRANCH**) to herot:

Go into the submodule directory (say GM/BATSRUS/) and type:

$ git push

Go back to the SWMF directory:

$ git push

*In conclusion, if you modified n submodules, you need to do n+1 adds and commits.*

## Working with stand-alone models (BATSRUS, GITM2, PWOM …)

Somewhat different from the SWMF, different Git repositories will be used in this case without the Git submodules (Git does not allow making the same repository submodules in different environments). Fortunately we only need the share and util repositories in addition to the model repository, so things are still not too complicated.

(1) First clone the BATSRUS repository (similar to **cvs checkout BATSRUS**):

$ git clone herot.engin.umich.edu:/GIT/FRAMEWORK/BATSRUS.git

(2) For sake of convenience, you can use Config.pl to get the util and share (or other necessary) repositories, for example

$ ./Config.pl -install -compiler=gfortran

will automatically clone (check out) the share and util repositories into BATSRUS. Note that these are not submodules (unlike in the SWMF), rather they are independent Git repositories.

We plan to develop some simple scripts to check the status and update, commit and push share/ and util/ from the main model directory. For now, the user has to go into these directories one-by-one. For example the following will get the status of the model including share and util:

$ git status; cd share; git status; cd ../util; git status; cd ..