# 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. We created several Git repositories (SWMF.git, BATSRUS.git, GITM2.git, share.git, util.git and so on). All the SWMF related Git repositories are stored in the

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

directory. To make cloning (downloading) the SWMF and/or individual models, like BATS-R-US, easier, the *Config.pl* scripts can take care of cloning the Git repositories corresponding to the various subdirectories.

## 3. Set up passwordless 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 Git Commands and Tools



Figure 1 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 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 SWMF repository with all models included will be about 900 MB instead of the 187 MB size of the CVS repository. Using --depth=NUMBER allows limiting the version history, for example:

$ git clone --depth=1 herot:/GIT/FRAMEWORK/SWMF 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.

To avoid typing the name of the Git server every time, it is convenient to define **gitclone** as an alias or function.

For csh and tcsh shells define a gitclone alias in the .cshrc file:

$ alias gitclone "git clone herot.engin.umich.edu:/GIT/FRAMEWORK/\!\*"

For sh, ksh, bash and zsh create a function in .profile, .bashrc, .zshrc or similar configuration file:

$ gitclone() { git clone herot.engin.umich.edu:/GIT/FRAMEWORK/"$@"; }

With this definition, any Git repository can be cloned easily, for example

$ giclone BATSRUS

$ gitclone SWMF --depth=1 SWMF\_NO\_HISTORY

### Checking the difference between the present git repository and the remote server (similar to **cvs -n update**)

First of all, it is useful to define a visual difference tool, such as tkdiff, to be accessible by Git. Use the following command

$ git config --global diff.tool tkdiff

When other developers modified the project and pushed changes to the remote repository, it is useful to check the differences between the local and remote repositories. We recommend the following workflow:

$ git fetch origin # download the remote branch to the local branch ‘origin’

$ git diff origin # compare the local branch with ‘origin’ (the newest version in remote)

$ git difftool origin # compare the local branch with origin using the difftool defined above

### Update the git repository from the remote server (similar to **cvs update**)

If the ‘origin’ branch was downloaded with ‘git fetch origin’, the changes can be merged with the local master branch using

$ git merge origin

Please refer to the internet for information about dealing with conflicts during the merging. If you are confident about the correctness of the remote repository, the fetch and merge steps can be replaced with a single command:

$ git pull

This 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

Check the changes in all the modified files:

$ git diff

$ git difftool

Also, you can use tkdiff directly 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

### Settings for line endings

If you're using Git to collaborate with others, ensure that Git is properly configured to handle line endings, on macOS use

$ git config --global core.autocrlf

Optionally, you can configure the way Git manages line endings on a per-repository basis by configuring a special  *.gitattributes* file. More information can be found here: <https://help.github.com/articles/dealing-with-line-endings>

## 5. Tools developed for the SWMF with many git repositories

The improved *Config.pl* scripts now checks the existence of the necessary components, for example, the share/ and util/ subdirectories, and clone them from herot if necessary. In addition, the *Config.pl* script in the SWMF can also clone the SWMF models (like BATSRUS, AMPS, etc) during installation.

We provide a script *gitall* in *share/Scripts.* This script will recursively search the git repositories and execute the git command passed to it. It is best to link or copy gitall into the execution path, so it can be used easily from any directory. For example, to check the status of all the git repositories in the SWMF type the following at the top-level directory:

$ gitall status

By default, there is an output only if there is a change in the repository. Using the -v flag (verbose)

$ gitall status -v

will show all the git repositories listed even if there are no changes. Other possible uses include

$ gitall fetch origin

$ gitall difftool origin

$ gitall merge origin

$ gitall pull

$ gitall push

Doing commit with gitall is not recommend since commit logs are typically independent in different repositories.

## 6. Using SWMF\_data and CRASH\_data

Just like with the CVS version, the large files are stored in the SWMF\_data and CRASH\_data repositories. These can be checked out into the home directory as

$ cd

$ gitclone SWMF\_data

$ gitclone CRASH\_data

This should be done before installing the SWMF or stand-alone model so that the data/ symbolic links can be properly created.

## 7. Working with the SWMF

Clone the core SWMF repository without models (similar to **cvs checkout SWMF\_core**):

$ gitclone SWMF

Now you have a local SWMF repository without share, util and the physics models. The new features of Config.pl can take care of cloning (downloading) the missing pieces:

$ ./Config.pl # show SWMF information and clone util and share if missing

$ ./Config.pl -install # install SWMF with all models (clone the entire SWMF)

$ ./Config.pl -install=BATSRUS,PWOM # install SWMF, clone GM/BATSRUS, PW/PWOM if missing

$ ./Config.pl -install=AMPS # reinstall SWMF, clone AMPS if necessary

*Note that models that are already present will not be cloned again during reinstallation.*

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

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

$ gitclone BATSRUS

Like in the SWMF, now you can use Config.pl to get all the necessary repositories, for example

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

will automatically clone (check out) the share, util and srcBATL repositories into BATSRUS if not yet present. Note that these are independent Git repositories.