SVN to Git guide for WRF developers

This is a guide for those with experience in Subversion (SVN) to get comfortable with Git commands. For instructions on opening pull requests and more Github-specific stuff, see the WRF Wiki on Github (you will need repository access; contact kavulich@ucar.edu or wrfhelp@ucar.edu for details).

One-time actions before doing anything else in Git....

Set global configuration options on your local machine(s)

Before you start experimenting with git, you should set the following two "global" git settings on the machine you're using.

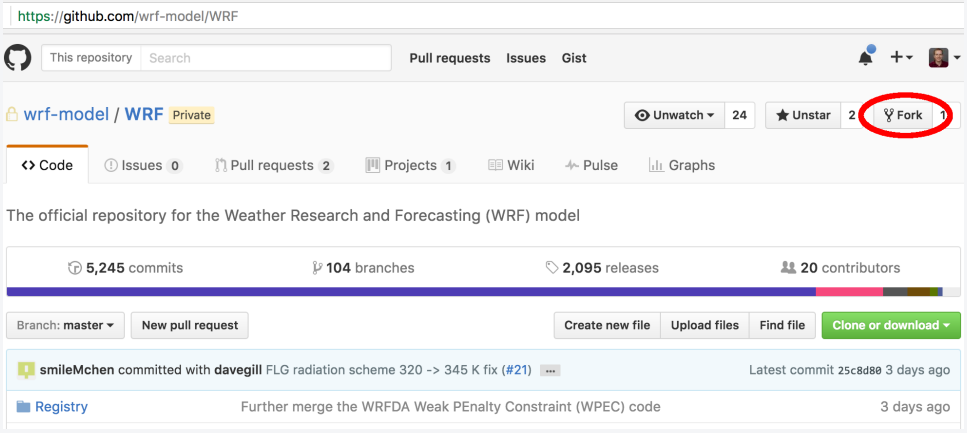
```
$ git config --global user.name "Your full name"
$ git config --global user.email your_email@ucar.edu
```

You should use the name and email that you would like to be seen in the repository logs.

If you're curious as to why we do this, this page has more information: https://git-scm.com/book/en/v2/Getting-Started-First-Time-Git-Setup

Create a new fork (Github-specific functionality)

One way of ensuring that your work will not interfere with others is to work off a fork. A fork is a "clone" (copy) of the main repository but stored under your own Github account. We recommend that development (outside of special cases which need extensive collaboration) be performed in forks under each developers' own Github account. Here you can create your own branches on your own copy of the repository, and they can be merged back to the "master" branch of the main repository through pull requests.



To fork the main repository, simply go to https://github.com/wrf-model/WRF in your browser (while logged in), and click "Fork" in the top-right corner. After the fork is created, you will be brought to the main webpage for your fork (https://www.github.com/username/WRF).

Checking out the code (creating a clone)

Old Procedure:	New Procedure:

`svn checkout https://svn-wrf-model.cgd.ucar.edu/trunk/`	Instead of checking out the main repository, in most cases you will check out your fork:
	`git clone https://username@github.com/username/WRF`
	On Mac machines you can probably omit the first "username" part, though it will depend on your git version:
	`git clone https://github.com/username/WRF`
	You can also set up your machine to connect via ssh; this is useful for scripting on Linux machines (such as Yellowstone/Cheyenne) to avoid having to enter your password every time you clone/push/pull code (Mac machines should have built-in password management).
	`git clone ssh://ssh@github.com/wrf-model/WRF`
	There are some instructions on how to set up an SSH key here: https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/
	Once it is generated, you can add an SSH key to your account here: https://github.com/settings/keys

Creating/checking out a branch

While it was only recommended that you do work on branches with SVN, ***with git all work should take place on branches***, since this allows you to access a lot of the new beneficial features that git and Github have to offer (including pull requests), and avoids accidentally committing directly to the main master branch (in Subversion this was called the "trunk").

Old procedure:	New procedure:

In subversion, branches are essentially just copies of the trunk stored in the SVN repository. They are created with the "svn copy" command:
 	
`svn copy https://svn-wrf-model.cgd.ucar.edu/trunk/ https://svn-wrf-model.cgd.ucar.edu/branches/new-branch`
	In Git, branches are way more involved, and way more useful. When you check out the code, you have access to all the repository's branches as well. Additionally, when you create a branch, it can be stored locally, rather than cluttering up the main repository, until you are ready to commit your changes to the master/trunk
	
	To create a new branch, simply use the "git checkout" command, with the "-b" argument to specify that you're creating a new branch
	
	`git checkout -b new-branch`
	
	If you forgot to create a new branch before starting your changes, that's no problem: the changes will automatically be carried over to your new branch with the above command.
	
	To see a list of branches that you have checked out locally, use the "git branch" command.
	
	`$ git branch`
	
	`  master`
	`* new-branch`
	
	To switch back to the master (or any other locally-tracked branch), use the "git checkout" command again, without the -b argument.
	
	`$ git checkout master`
	
	`Switched to branch 'master'`
	`Your branch is up-to-date with 'origin/master'.`
	
	As I hinted at above, the above command only shows you "local" branches, that you have created or checked out from the "remote" repository on Github. To see a full list of branches you could check out, use git branch -a
	
	`$ git branch -a`
	
	`  master`
	`* new-branch`
	`  remotes/origin/AER-radiation`
	`  remotes/origin/AFWA_SAMPLE_BRANCH`
	`  remotes/origin/AFWA_test_7242`
	`  remotes/origin/ANL_BDY`
	`  ...`
	
	All these listings under "remotes/origin" are "remote" branches, which in this case are from the "origin" of your repository, which is the Github repository.
	
	To check out one of these "remote" branches, again use the "git checkout" command.
	
	`$ git checkout AER-radiation`
	
	`Branch AER-radiation set up to track remote branch AER-radiation from origin.`
	`Switched to a new branch 'AER-radiation'`
	
	You are now tracking the remote branch "AER-radiation" locally, and can make changes to it and push those changes back to that branch in the main repository.

Adding, moving, and deleting files

Old procedure:	New procedure:

In Subversion, you must use specific commands to tell SVN that you are adding, renaming, or deleting a file:
 	
`$ svn add new_file.txt`
`$ svn mv README README.NEW`
`$ svn rm README.hydro`
	The same is true in Git, and the commands are all the same.
	
	`$ git add new_file.txt`
	`$ git mv README README.NEW`
	`$ git rm README.hydro`

There is one more catch to using git: If you change a file it is not automatically added to the list of files to commit. Read the next sections for more info.

Viewing your uncommitted changes

Status (list of changed files)

Old procedure:	New procedure: **In git, the command is the same, but it gives you more information:**

In SVN, to see the list of files you have modified, you use the "svn status" command
 	
`$ svn status`
 	
`?  un-added_file.txt`
`D  README`
`  > moved to README.NEW`
`A  +  README.NEW`
`  > moved from README`
`D  README.hydro`
`A  new_file.txt`
	`$ git status`
	
	`# On branch new-branch`
	`# Changes to be committed:`
	`#  (use "git reset HEAD <file>..." to unstage)`
	`#`
	`#  renamed:  README -> README.NEW`
	`#  deleted:  README.hydro`
	`#  new file:  new_file.txt`
	`#`
	`# Changed but not updated:`
	`#  (use "git add <file>..." to update what will be committed)`
	`#  (use "git checkout -- <file>..." to discard changes in working directory)`
	`#`
	`#  modified:  var/build/depend.txt`
	`#`
	`# Untracked files:`
	`#  (use "git add <file>..." to include in what will be committed)`
	`#`
	`#  un-added_file.txt`
	
	Take special note of the second section: "Changed but not updated". These are existing files which you have changed, but you have not told git that you want to commit changes to these files yet. To *stage* these changes (tell git you want to commit them), use the "git add" command:
	
	`$ git add var/build/depend.txt`
	
	`$ git status`
	
	`# On branch new-branch`
	`# Changes to be committed:`
	`#  (use "git reset HEAD <file>..." to unstage)`
	`#`
	`#  renamed:  README -> README.NEW`
	`#  deleted:  README.hydro`
	`#  new file:  new_file.txt`
	`#  modified:  var/build/depend.txt`
	`#`
	`# Untracked files:`
	`#  (use "git add <file>..." to include in what will be committed)`
	`#`
	`#  un-added_file.txt`

Diff (Full list of changes line-by-line)

Old procedure:	New procedure: **In git, the command is the same, but because of the different commit procedure there are extra stipulations**

In SVN, to see the modifications you have made, you use the "svn diff" command
 	
`$ svn diff`
 	
Index: var/da/da_setup_structures /da_setup_be_ncep_gfs.inc
 	
`===================================================================`
 	
--- var/da/da_setup_structures /da_setup_be_ncep_gfs.inc (revision 9440)
 	
+++ var/da/da_setup_structures /da_setup_be_ncep_gfs.inc (working copy)
 	
@@ -182,7 +182,7 @@
 	
ALLOCATE (be % wgvz(its:ite,jts:jte,kts:kte))
 	
!
 	
-! 2, load the WFR model latitude and map factor:
 	
+! 2, load the WRF model latitude and map factor:
 	
#ifdef DM_PARALLEL
 	
ij = (ide- ids+1)*(jde- jds+1)
 	
to see the difference between two revisions, use "-r"
 	
svn diff -r9200:9400
 	
svn diff -r9000:HEAD
 	
to see the difference between two branches, specify the whole URL
 	
svn diff https://svn-wrf-model.cgd.ucar.edu /branches/
	To see the difference between your modifications and the checked out code, simply use git diff
	
	`[kavulich@yslogin3 WRF]$ git diff`
	
	`diff --git a/var/da/da_setup_structures/da_setup_be_ncep_gfs.inc b/var/da /da_setup_structures/da_setup_be_ncep_gfs.inc`
	
	`index 50e9f92..4133795 100644`
	
	`--- a/var/da/da_setup_structures/da_setup_be_ncep_gfs.inc`
	
	`+++ b/var/da/da_setup_structures/da_setup_be_ncep_gfs.inc`
	
	`@@ -182,7 +182,7 @@ subroutine da_setup_be_ncep_gfs( grid, be )`
	
	`ALLOCATE ( be % wgvz(its:ite,jts:jte,kts:kte) )`
	
	`!`
	
	`-! 2, load the WFR model latitude and map factor:`
	
	`+! 2, load the WRF model latitude and map factor:`
	
	`#ifdef DM_PARALLEL`
	
	`ij = ( ide- ids+1)*( jde- jds+1)`
	
	To see the difference for files already staged with "git add" (but not committed), use the "–staged" option
	
	git diff --staged
	
	To see the difference between your changes and a specific commit, simply specify that commit's hash. You can also use the shortened hash; the first two commands here are equivalent:
	
	`$ git diff ebba289b974e4ce33e959737d75edf28ce6a2558 dea56d515ebdeb73170030ddee60270b185f14b5`
	
	`$ git diff ebba289 dea56d5`
	
	You can also specify "HEAD" for the head (most recent) revision.
	
	`$ git diff HEAD dea56d5`
	
	To see the difference between two branches, specify the two branch names in the following way:
	
	`$ git diff branch_1..branch_2`

Reverting unwanted changes

Old procedure:	New procedure:

In Subversion, to reset a changed file to the state it was in when you checked it out, you use the "svn revert" command
 	
`$ svn revert README.hydro`
 	
`Reverted 'README.hydro'`
	In git, the command will be different depending on what kind of change you want to revert. You may have noticed some of these commands in the "git status" dialog above.
	
	To un-stage a file (tell git you don't want to commit a file, but keep the changes locally), use "git reset HEAD":
	
	`$ git reset HEAD var/build/depend.txt`
	
	`# On branch new-branch`
	`# Changes to be committed:`
	`#   (use "git reset HEAD <file>..." to unstage)`
	`#`
	`#       renamed:    README -> README.NEW`
	`#       deleted:    README.hydro`
	`#       new file:   new_file.txt`
	`#`
	`# Changed but not updated:`
	`#   (use "git add <file>..." to update what will be committed)`
	`#   (use "git checkout -- <file>..." to discard changes in working directory)`
	`#`
	`#       modified:   var/build/depend.txt`
	`#`
	`# Untracked files:`
	`#   (use "git add <file>..." to include in what will be committed)`
	`#`
	`#       un-added_file.txt`
	
	To revert all changes to a file, use "git checkout":
	
	`$ git checkout var/build/depend.txt`
	
	`# On branch new-branch`
	`# Changes to be committed:`
	`#   (use "git reset HEAD <file>..." to unstage)`
	`#`
	`#       renamed:    README -> README.NEW`
	`#       deleted:    README.hydro`
	`#       new file:   new_file.txt`
	`#`
	`# Untracked files:`
	`#   (use "git add <file>..." to include in what will be committed)`
	`#`
	`#       un-added_file.txt`
	
	To revert all changes you've made to your local working branch, use "git reset --hard". **Use this option with caution: it discards *everything* except for untracked files!**
	
	`$ git reset --hard`
	
	`HEAD is now at 09534a2 TYPE: bug fix`
	
	`$ git status`
	
	`# On branch new-branch`
	`# Untracked files:`
	`#   (use "git add <file>..." to include in what will be committed)`
	`#`
	`#       new_file.txt`
	`#       un-added_file.txt`
	`nothing added to commit but untracked files present (use "git add" to track)`

Committing your changes

Old procedure:	New procedure:

In Subversion, committing was always done back to the main repository, and done with the "svn commit" command:
 	
`svn commit -m 'This is my commit message'`
`Sending testdata.txt`
`Transmitting file data .`
`Committed revision 287.`
	In Git, you can commit "locally" as many times as you want. This means that you can keep your changes under version control before they are ready to be put in the trunk ("master").
	
	`git commit -m 'This is my commit message'`
	`[new-branch a4cf86a] This is my commit message`
	`1 files changed, 1 insertions(+), 1 deletions(-)`
	
	Once your changes are ready for the trunk/master, you can push your local branch into the main repository ***(remember: always work on a branch in git!)***. In this case "new-branch" is the branch you are on, having created it in a previous step.
	
	`$ git push -u origin new-branch`
	`Password:`
	`Counting objects: 4, done.`
	`Delta compression using up to 32 threads.`
	`Compressing objects: 100% (3/3), done.`
	`Writing objects: 100% (3/3), 5.51 KiB, done.`
	`Total 3 (delta 1), reused 0 (delta 0)`
	`To https://mkavulich@github.com/wrf-model/WRF`
	`09534a2..d94c5b7 new-branch -> new-branch`
	`Branch new-branch set up to track remote branch new-branch from origin.`

From here, you can submit a pull request through Github, which will be the official proposal to put your changes into the master/trunk.

Viewing the change log

Old procedure:	New procedure:

In SVN, the only command for viewing the commit log is "svn log". This is commonly used in 3 different ways:
 	
`svn log (displays log of current checked-out code)`
 	
`svn log -v (displays verbose log, including files changed in each commit)`
 	
`svn log -r9000:HEAD https://svn-wrf-model.cgd.ucar.edu/trunk/`
`(displays log of specified URL from revision 9000 to the current revision)`
	For Git, the above three commands work a bit differently. There is still your regular old "log" command:
	
	`git log (displays log of current checked-out code, including local commits).`
	
	`git whatchanged (displays verbose log, including files changed in each commit)`

Viewing "who's to blame" for past commits

Old procedure:	New procedure:

In SVN, "svn blame file.txt" would give you the line-by-line contents of "file.txt", prepended with information about the author and revision the last time each line was changed.
 	
`svn blame file.txt`
 	
If you'd like to see the log entry for the commit which changed a certain line, you can use svn log and specify the revision number:
 	
`svn log -r8250`
	Git has an analogous function: "git blame" which gives you the author information, date/time, and SHA hash of last commit.
	
	`git blame file.txt`
	
	If you'd like to see the log entry for the commit which changed a certain line, you can similarly use git log and specify the SHA hash:
	
	`git log 92e6ba8`

New github goodies:

Github also has detailed "blame" functionality built in: Check this out! https://help.github.com/articles/using-git-blame-to-trace-changes-in-a-file/