GIT 101 – Moving from SVN to GIT. - FTRUJILLO

https://github.s3.amazonaws.com/media/progit.en.pdf This is the complete reference you need
http://git-scm.com/docs
https://git.wiki.kernel.org/index.php/Main_Page

LINUX Install binary using yum.

```
$ yum install curl-devel expat-devel gettext-devel \
openssl-devel zlib-devel

$ yum install git-core
```

OR Install from source

```
$ wget http://git-scm.com/download
$ tar -zxf git-1.6.0.5.tar.gz
$ cd git-1.6.0.5
$ make prefix=/usr/local all
$ sudo make prefix=/usr/local install
```

WINDOWS Install

I have a section towards the end of this document to show Windows install

```
nsglnxdev1.micron.com:/home/ftrujillo [643]$ cat .gitignore
*~
.snapshot/
*.pyc
*.[oa]
log/*.log
tmp/sockets/
tmp/sessions/
tmp/cache/
tmp/pids/
.svn/
nbproject/private/
dist/
build/
*.class
*.db
```

Initial GLOBAL one time setup. [creates ~/.gitconfig]

Uses name and email only for giving credit to checkins.

```
$ git config --global user.name "Francis Trujillo"
$ git config --global user.email ftrujillo@micron.com
$ git config --global core.editor emacs
```

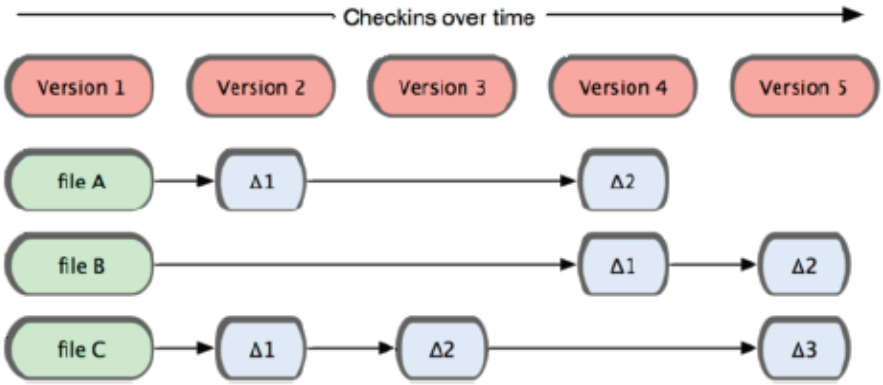
LINUX - tell Git to convert CRLF to LF on commit but not the other way around

```
$ git config --global core.autocrlf input

$ git config --global merge.tool kdiff3
$ git config --global diff.tool kdiff3
$ git config --global difftool.kdiff3.cmd 'kdiff3 $LOCAL $REMOTE'
$ git config --global difftool.prompt false
$ git config --global core.excludesfile ~/.gitignore
$ git config –list
```

Snapshots, Not Differences

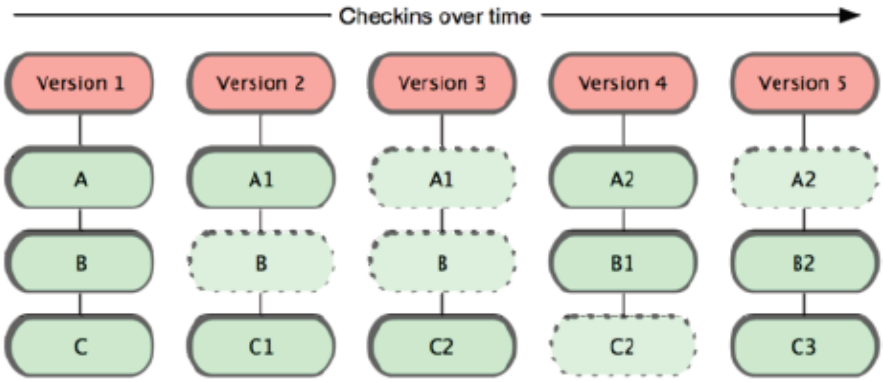
The major difference between Git and any other VCS (Subversion and friends included) is the way Git thinks about its data. Conceptually, most other systems store information as a list of file-based changes.

Figure 1.4: Other systems tend to store data as changes to a base version of each file.

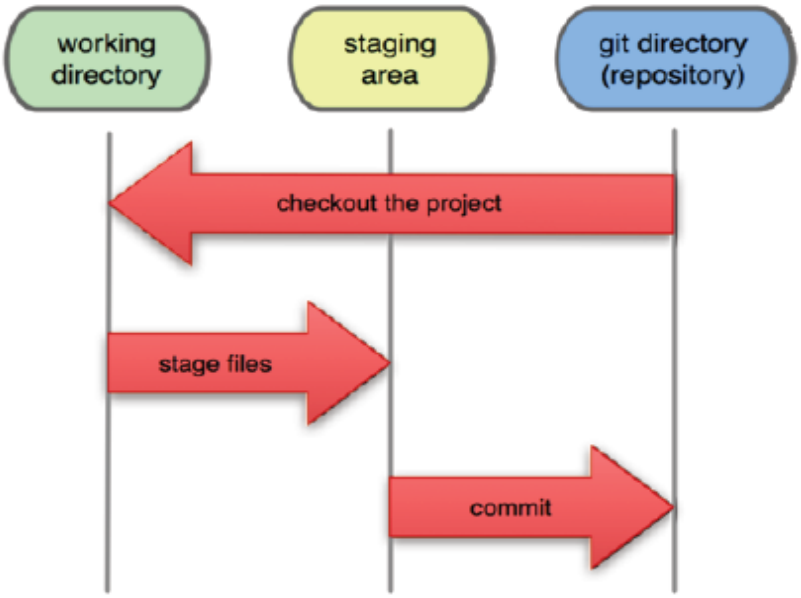


Git doesn't think of or store its data this way. Instead, Git thinks of its data more like a set of snapshots of a mini filesystem. Every time you commit, or save the state of your project in Git, it basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn't store the file again—just a link to the previous identical file it has already stored. Git thinks about its data more like Figure 1.5.

Figure 1.5: Git stores data as snapshots of the project over time.



This is an important distinction between Git and nearly all other VCSs. It makes Git reconsider almost every aspect of version control that most other systems copied from the previous generation. This makes Git more like a mini filesystem with some incredibly powerful tools built on top of it, rather than simply a VCS. We'll explore some of the benefits you gain by thinking of your data this way when we cover Git branching in Chapter 3.



The Git directory is where Git stores the metadata and object database for your project. This is the most important part of Git, and it is what is copied when you clone a repository from another computer.

The working directory is a single checkout of one version of the project. These files are pulled out of the compressed database in the Git directory and placed on disk for you to use or modify.

The staging area is a simple file, generally contained in your Git directory, that stores information about what will go into your next commit. It's sometimes referred to as the index, but it's becoming standard to refer to it as the staging area.

The basic Git workflow goes something like this:

1. You modify files in your working directory.
2. You stage the files, adding snapshots of them to your staging area.
3. You do a commit, which takes the files as they are in the staging area and stores that snapshot permanently to your Git directory.

If a particular version of a file is in the git directory, it's considered committed. If it's modified but has been added to the staging area, it is staged. And if it was changed since it was checked out but has not been staged, it is modified. In Chapter 2, you'll learn more about these states and how you can either take advantage of them or skip the staged part entirely.

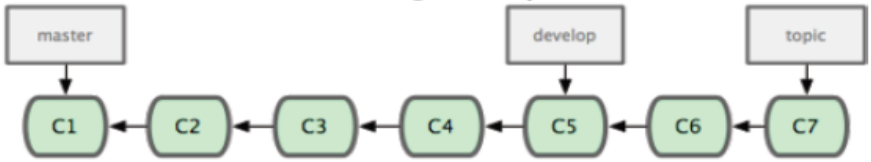
3.4.1 Long-Running Branches

Because Git uses a simple three-way merge, merging from one branch into another multiple times over a long period is generally easy to do. This means you can have several branches that are always open and that you use for different stages of your development cycle; you can merge regularly from some of them into others.

Many Git developers have a workflow that embraces this approach, such as having only code that is entirely stable in their `master` branch — possibly only code that has been or will be released. They have another parallel branch named develop or next that they work from or use to test stability — it isn't necessarily always stable, but whenever it gets to a stable state, it can be merged into `master`. It's used to pull in topic branches (short-lived branches, like your earlier `iss53` branch) when they're ready, to make sure they pass all the tests and don't introduce bugs.

In reality, we're talking about pointers moving up the line of commits you're making. The stable branches are farther down the line in your commit history, and the bleeding-edge branches are farther up the history (see Figure 3.18).

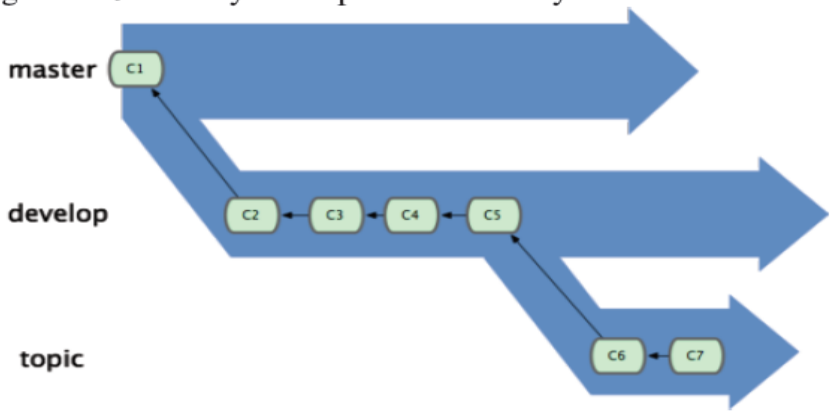
Figure 3.18: More stable branches are generally farther down the commit history.



It's generally easier to think about them as work silos, where sets of commits graduate to a more stable silo when they're fully tested (see Figure 3.19).

You can keep doing this for several levels of stability. Some larger projects also have a `proposed` or `pu` (proposed updates) branch that has integrated branches that may not be ready to go into the `next` or `master` branch. The idea is that your branches are at various levels of stability; when they reach a more stable level, they're merged into the

Figure 3.19: It may be helpful to think of your branches as silos.



branch above them. Again, having multiple long-running branches isn't necessary, but it's often helpful, especially when you're dealing with very large or complex projects.

Optional GLOBAL setup

```
$ echo '*.png diff=exif' >> .gitattributes
$ git config diff.exif.textconv exiftool

$ echo ' *.doc.* binary diff=word' >> .gitattributes
$ git config diff.word.textconv strings
```

```
$ cat ~/.gitattributes
*.xml* binary -crlf -diff
*.doc.* binary diff=word
```

Manually create this file.

```
$ cat ~/.gitignore
*~
.snapshot/
.svn/
*.[oa]
log/*.log
tmp/sockets/
tmp/sessions/
tmp/cache/
tmp/pids/
nbproject/private/
nbproject/private/private.properties
dist/
build/
*.class
*.db
```

GLOBAL Git Aliases that you can setup or use

```
$ git config --global alias.new              'init .'
$ git config --global alias.import           'add .'
$ git config --global alias.ci               'commit'

# git branch                                 (no args shows all branches)
# git branch -v                     (shows all branches with last commit)
# git branch --merged               (show branches merged into current)
# git branch  --no-merged           (show unmerged branches into current)
# git branch -D BRANCH              (Force DELETE branch!! Use caution!)


$ git config --global alias.br               'branch'              BRANCH
$ git config --global alias.co               'checkout'            BRANCH
$ git config --global alias.brco             'checkout -b'         BRANCH
$ git config --global alias.mg               'merge'               BRANCH
# After merge, then delete branch using -d
$ git config --global alias.brrm             'branch -d'           BRANCH

$ git config --global alias.st               'status'
$ git config --global alias.l                'log --pretty=format:"%h %d| %an | %cn | %cd
| %cr | %s" -100'
$ git config --global alias.ll               'log --pretty=format:"%h %d| %s" –graph -
100'

$ git config --global alias.dt               'difftool'
$ git config --global alias.mt               'mergetool'


$ git config --global alias.unstage          'reset HEAD'
$ git config --global alias.revert           'checkout --'     FILE

$ git config --global alias.remadd           'remote add'      NAME URL
$ git config --global alias.remdownload      'fetch'           NAME
$ git config --global alias.remupdate        'pull'            NAME BRANCH
$ git config --global alias.remupload        'push'            NAME BRANCH
$ git config --global alias.remshow          'remote -v'
$ git config --global alias.remrm            'remote rm'

$ git config --global alias.last             'log -1 HEAD'
```

So, `git fetch origin` fetches any new work that has been pushed to that server since you cloned (or last fetched from) it. It's important to note that the `fetch` command **pulls the data to your local repository — it doesn't automatically merge it with any of your work or modify what you're currently working on.** You have to merge it manually into your work when you're ready.

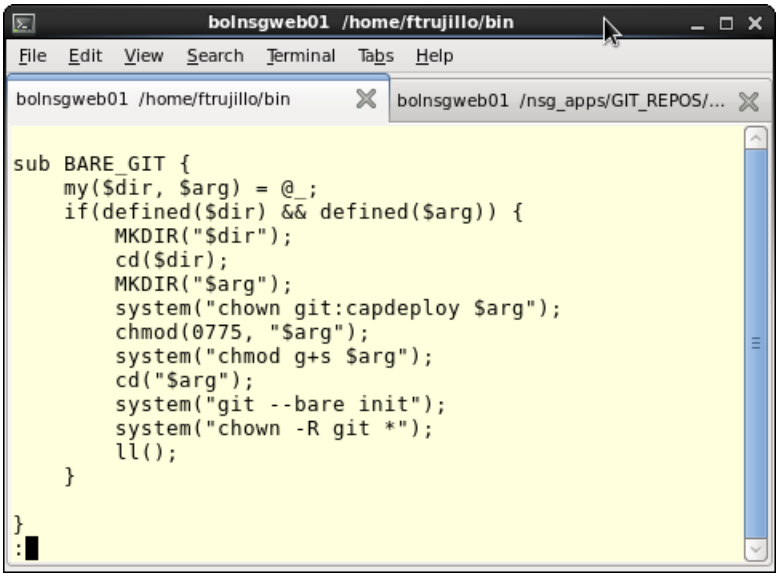
Running `git pull` generally **fetches data** from the server you originally cloned from **and automatically tries to merge it** into the code you're currently working on.

The HEAD

```
$ cat .git/HEAD
ref: refs/heads/master
```

Bare Repository on a Server

BARE_GIT("/nsg_apps/GIT_REPOS/integration_mfg", "learning.git");



A remote repository is generally a bare repository — a Git repository that has no working directory. Because the repository is only used as a collaboration point, there is no reason to have a snapshot checked out on disk; it's just the Git data. In the simplest terms, a bare repository is the contents of your project's .git directory and nothing else.

In order to initially set up any Git server, you have to export an existing repository into a new bare repository — a repository that doesn't contain a working directory. This is generally straightforward to do. In order to clone your repository to create a new bare repository, you run the clone command with the --bare option. By convention, bare repository directories end in .git, like so:

```
$ scp -r my_project.git user@git.example.com:/opt/git
$ git clone user@git.example.com:/opt/git/my_project.git
$ ssh user@git.example.com
$ cd /opt/git/my_project.git            USERS must have READ/WRITE to this dir.
$ git init --bare –shared
```

Local Protocol

```
$ git clone /opt/git/project.git
$ git clone file:///opt/git/project.git
$ git remote add local_proj /opt/git/project.git   Add to an existing Git project
```

Secure Shell (SSH)

```
$ git clone ssh://user@server:project.git
$ git clone user@server:project.git
```

```
$ sudo adduser git
$ su git
$ cd
$ mkdir .ssh

$ cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
$ cat /tmp/id_rsa.josie.pub >> ~/.ssh/authorized_keys
$ cat /tmp/id_rsa.jessica.pub >> ~/.ssh/authorized_keys

$ sudo vim /etc/passwd
    git:x:1000:1000::/home/git:/usr/bin/git-shell
```

```
$ cd /opt/git
$ mkdir project.git
$ cd project.git
$ git --bare init

# on Johns computer
$ cd myproject
$ git init
$ git add .
$ git commit -m 'initial commit'
$ git remote add origin git@gitserver:/opt/git/project.git
$ git push origin master

# someone else.
$ git clone git@gitserver:/opt/git/project.git
$ vim README
$ git commit -am 'fix for the README file'
$ git push origin master
```

HTTP/S Protocol

```
$ cd /var/www/htdocs/
$ git clone --bare /path/to/git_project gitproject.git
$ cd gitproject.git
$ mv hooks/post-update.sample hooks/post-update
$ chmod a+x hooks/post-update
```

The post-update hook that comes with Git by default runs the appropriate command (git update-server-info) to make HTTP fetching and cloning work properly. This command is run when you **push to this repository over SSH**

```
$ git clone http://example.com/gitproject.git
```

It's possible to make Git push over HTTP as well, although that technique isn't as widely used and requires you to set up complex WebDAV requirements.

Another nice thing is that HTTP is such a commonly used protocol that corporate firewalls are often set up to allow traffic through this port.

Git on a Server

In order to initially set up any Git server, you have to export an existing repository into a new bare repository

```
$ git clone --bare my_project my_project.git
```

Git Logging

```
$ git log
$ git log -p -2
$ git log –stat
$ git log --pretty=oneline
$ git log --pretty=format:"%h - %an, %ar : %s"
$ git log --pretty=format:"%h %s" --graph
$ git log --since=2.weeks
$ git checkout -- benchmarks.rb
```

REVERT

Option	Description of Output
%H	Commit hash
%h	Abbreviated commit hash
%T	Tree hash
%t	Abbreviated tree hash
%P	Parent hashes
%p	Abbreviated parent hashes
%an	Author name
%ae	Author e-mail
%ad	Author date (format respects the date= option)
%ar	Author date, relative
%cn	Committer name
%ce	Committer email
%cd	Committer date
%cr	Committer date, relative
%s	Subject

Git

```
$ git init
$ git add .
$ git commit –m ”initial commit”
$ git status
$ git rm grit.gemspec
$ git rm log/\*.log
$ git mv file_from  file_to
$ git clone ************
```

```
nsglnxdev1.micron.com /home/ftrujillo/NetBeansProjects/test
File Edit View Search Terminal Tabs Help
nsglnxdev1.micron.com /home/ftrujillo/N...   telinux57.micron.com /u/ftrujillo/bin   telinux11.micron.com /u/ftrujillo
nsglnxdev1.micron.com:/home/ftrujillo/NetBeansProjects [1659]$ mkdir test
nsglnxdev1.micron.com:/home/ftrujillo/NetBeansProjects [1660]$ cd test
nsglnxdev1.micron.com:/home/ftrujillo/NetBeansProjects/test [1661]$ git init .
Initialized empty Git repository in /home/ftrujillo/NetBeansProjects/test/.git/
nsglnxdev1.micron.com:/home/ftrujillo/NetBeansProjects/test [1662]$ tree --charset=ISO-8859 -pags $PWD
/home/ftrujillo/NetBeansProjects/test
`-- [drwxr-xr-x ftrujill        4096]  .git
    |-- [drwxr-xr-x ftrujill        4096]  branches
    |-- [-rw-r--r-- ftrujill          92]  config
    |-- [-rw-r--r-- ftrujill          73]  description
    |-- [-rw-r--r-- ftrujill          23]  HEAD
    |-- [drwxr-xr-x ftrujill        4096]  hooks
    |   |-- [-rwxr-xr-x ftrujill         452]  applypatch-msg.sample
    |   |-- [-rwxr-xr-x ftrujill         896]  commit-msg.sample
    |   |-- [-rwxr-xr-x ftrujill         160]  post-commit.sample
    |   |-- [-rwxr-xr-x ftrujill         548]  post-receive.sample
    |   |-- [-rwxr-xr-x ftrujill         189]  post-update.sample
    |   |-- [-rwxr-xr-x ftrujill         398]  pre-applypatch.sample
    |   |-- [-rwxr-xr-x ftrujill        1578]  pre-commit.sample
    |   |-- [-rwxr-xr-x ftrujill        1239]  prepare-commit-msg.sample
    |   |-- [-rwxr-xr-x ftrujill        4951]  pre-rebase.sample
    |   `-- [-rwxr-xr-x ftrujill        3611]  update.sample
    |-- [drwxr-xr-x ftrujill        4096]  info
    |   `-- [-rw-r--r-- ftrujill         240]  exclude
    |-- [drwxr-xr-x ftrujill        4096]  objects
    |   |-- [drwxr-xr-x ftrujill        4096]  info
    |   `-- [drwxr-xr-x ftrujill        4096]  pack
    `-- [drwxr-xr-x ftrujill        4096]  refs
        |-- [drwxr-xr-x ftrujill        4096]  heads
        `-- [drwxr-xr-x ftrujill        4096]  tags

10 directories, 14 files
nsglnxdev1.micron.com:/home/ftrujillo/NetBeansProjects/test [1663]$
```

Adding Remote Repositories

```
$ git remote add pb git://github.com/paulboone/ticgit.git
$ git remote –v
$ git fetch [remote-name]
$ git fetch origin
$ git pull  *******
$ git push origin master
$ git remote show origin
$ git remote rename pb paul
$ git remote rm paul
```

Tagging

```
$ git tag
$ git tag -l 'v1.4.2.*'
$ git tag -a v1.4 -m 'my version 1.4'
$ git show v1.4
```

```
$ git log --pretty=oneline
15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'
a6b4c97498bd301d84096da251c98a07c7723e65 beginning write support
0d52aaab4479697da7686c15f77a3d64d9165190 one more thing
6d52a271eda8725415634dd79daabbc4d9b6008e Merge branch 'experiment'
0b7434d86859cc7b8c3d5e1dddfed66ff742fcbc added a commit function
```

```
4682c3261057305bdd616e23b64b0857d832627b added a todo file
166ae0c4d3f420721acbb115cc33848dfcc2121a started write support
9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile
964f16d36dfccde844893cac5b347e7b3d44abbc commit the todo
8a5cbc430f1a9c3d00faaeffd07798508422908a updated readme
```

```
$ git tag -a v1.2 9fceb02
```

By default, the git push command doesn't transfer tags to remote servers. You will have to explicitly push tags to a shared server after you have created them. This process is just like sharing remote branches you can run git push origin [tagname].

```
$ git push origin v1.5                    TRANSFERS ONLY THIS TAG TO SERVER
$ git push origin --tags                  TRANSFERS ALL TAGS TO SERVER
```

git clone will give you the whole repository.
After the clone, you can list the tags with `git tag -l` and then checkout a specific tag: `git checkout tags/<tag_name>`

Git Branching

The default branch name in Git is master.
How does Git know what branch you're currently on? It keeps a special pointer called HEAD.

```
$ git branch              Run it with no arguments, you get a simple listing of your current branches
iss53
* master
testing

$ git branch -v
$ git checkout master
$ git status
$ git branch testing      The git branch command only created a new branch— it didn't switch to that branch
$ git checkout testing    This moves HEAD to point to the testing branch
$ git checkout master     flip back to master
$ git merge testing       then merge changes
$ git branch -d testing   then delete branch if no longer needed.
$ git checkout iss53      Switch to another branch (which does not have the changes pulled in yet)
$ git merge master        Pull changes into iss53 or wait until you merge iss53 back to master
$ git mergetool           If conflicts occur.
$ git commit –m "fixed conflict"
```

NOTE: that if your working directory or staging area has uncommitted changes that conflict with the branch you're checking out, Git won't let you switch branches. It's best to have a clean working state when you switch branches.

```
$ git branch --merged
iss53
* master

$ git branch --no-merged      To see all the branches that contain work you haven't yet merged in.
testing                       trying to delete it with git branch -d will fail
```

NOTE: If you are sure you want to delete it, run **git branch -D** testing. If you really do want to **delete the branch and lose that work**, you can force it with -D

Pushing

When you want to share a branch with the world, you need to push it up to a remote

that you have write access to. Your local branches aren't automatically synchronized to the remotes you write to.

```
git push (remote) (branch)
```

```
$ git push origin serverfix
```
"Take my serverfix and make it the remote's serverfix."

Or if want name different on remote

```
$ git push origin serverfix:awesomebranch
```

```
$ git fetch origin
```

NOTE: It's important to note that when you do a fetch that brings down new remote branches, you don't automatically have local, editable copies of them.

```
$ git merge  origin/serverfix
```
To merge this work into your current working branch

NOTE: If you want your own serverfix branch that you can work on, you can base it off your remote branch

```
$ git checkout -b serverfix origin/serverfix
```

Tracking Branches

```
$ git checkout --track origin/serverfix
$ git checkout -b sf origin/serverfix
```

Deleting Remote Branches

git push [remotename] :[branch]

You and your collaborators are finished with a feature and have merged it into your remote's master branch.

$ git push origin :serverfix

Creating TARBALL archive.

```
$ git archive master --prefix='project/' | gzip > `git describe master`.tar.gz
```

You want to see what is in your experiment branch that hasn't yet been merged into your master branch. You can ask Git to show you a log of just those commits with master..experiment — that means "all commits reachable by experiment that aren't reachable by master."

```
$ git log master..experiment
```

File Annotation

```
$ git blame -L 12,22 simplegit.rb
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 12) def show(tree = 'master')
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 13) command("git show #{tree}")
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 14) end
^4832fe2 (Scott Chacon 2008-03-15 10:31:28 -0700 15)
9f6560e4 (Scott Chacon 2008-03-17 21:52:20 -0700 16) def log(tree = 'master')
79eaf55d (Scott Chacon 2008-04-06 10:15:08 -0700 17) command("git log #{tree}")
9f6560e4 (Scott Chacon 2008-03-17 21:52:20 -0700 18) end
9f6560e4 (Scott Chacon 2008-03-17 21:52:20 -0700 19)
42cf2861 (Magnus Chacon 2008-04-13 10:45:01 -0700 20) def blame(path)
42cf2861 (Magnus Chacon 2008-04-13 10:45:01 -0700 21) command("git blame #{path}")
42cf2861 (Magnus Chacon 2008-04-13 10:45:01 -0700 22) end
```

```
$ git blame -C -L 141,153 GITPackUpload.m
f344f58d GITServerHandler.m (Scott 2009-01-04 141)
f344f58d GITServerHandler.m (Scott 2009-01-04 142) - (void) gatherObjectShasFromC
f344f58d GITServerHandler.m (Scott 2009-01-04 143) {
70befddd GITServerHandler.m (Scott 2009-03-22 144) //NSLog(@"GATHER COMMI
ad11ac80 GITPackUpload.m (Scott 2009-03-24 145)
ad11ac80 GITPackUpload.m (Scott 2009-03-24 146) NSString *parentSha;
ad11ac80 GITPackUpload.m (Scott 2009-03-24 147) GITCommit *commit = [g
ad11ac80 GITPackUpload.m (Scott 2009-03-24 148)
ad11ac80 GITPackUpload.m (Scott 2009-03-24 149) //NSLog(@"GATHER COMMI
ad11ac80 GITPackUpload.m (Scott 2009-03-24 150)
56ef2caf GITServerHandler.m (Scott 2009-01-05 151) if(commit) {
56ef2caf GITServerHandler.m (Scott 2009-01-05 152) [refDict setOb
56ef2caf GITServerHandler.m (Scott 2009-01-05 153)
```

Subversion to Git

Create a file called users.txt
that has this mapping in a format like this:

> schacon = Scott Chacon <schacon@geemail.com>
> selse = Someo Nelse selse@geemail.com

```
$ svn log --xml | grep author | sort -u | perl -pe 's/.>(.?)<./$1 = /'  > users.txt

$ git-svn clone http://my-project.googlecode.com/svn/ \
--authors-file=users.txt --no-metadata -s my_project

$ cp -Rf .git/refs/remotes/tags/* .git/refs/tags/
$ rm -Rf .git/refs/remotes/tags

$ cp -Rf .git/refs/remotes/* .git/refs/heads/
$ rm -Rf .git/refs/remotes

$ git push origin –all
```

GUI's – undetermined right now. I am command line freak.

https://netbeans.org/kb/docs/ide/git.html

http://git-scm.com/downloads/guis

- **gitk** - graphical history browser, in Tcl/Tk, distributed with Git (usually in `gitk` package)
- **git gui** - graphical commit tool, in Tcl/Tk, distributed with Git (usually in `git-gui` package)
- **QGit** - uses Qt toolkit
- **Giggle** - uses GTK+ toolkit
- **git-cola** - uses PyQt4
- **gitg** - GTK+/GNOME clone of GitX
- **tig** - **t**ext mode **i**nterface for **g**it, is GUI and pager, uses ncurses
- **GitForce** Git tool with Graphical user interface, available under GNU GPL license

http://akyl.net/how-install-latest-version-git-centos-63

http://www.maketecheasier.com/6-useful-graphical-git-client-for-linux/

PROXY

Ubuntu Bash:
http_proxy='http://proxyb.micron.com:8080'
export http_proxy
fixes the irritating Ubuntu Menu Proxy warning
alias gvim='UBUNTU_MENUPROXY= gvim'

CSH / TCSH:

nsglnxdev1.micron.com:/home/ftrujillo [109]$ grep http_proxy ~/.cshrc
setenv http_proxy "proxy.micron.com:8080"

Windows KDIFF3 Install

http://kdiff3.sourceforge.net/

Click on download link, then on link that points to something like this.

http://hivelocity.dl.sourceforge.net/project/kdiff3/kdiff3/0.9.97/KDiff3-32bit-Setup_0.9.97.exe

You will have to add this to your PATH manually.

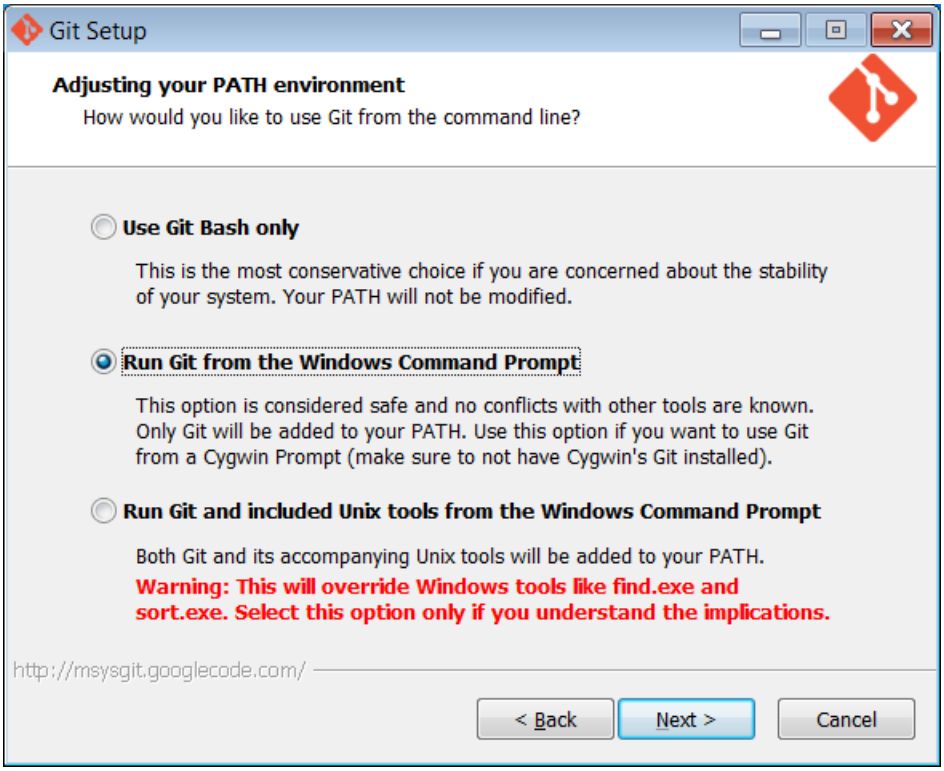
C:\Program Files (x86)\KDiff3
Not C:\Program Files (x86)\KDiff3\bin

```
C:\Program Files (x86)\KDiff3\bin>dir
 Volume in drive C is OS
 Volume Serial Number is FA36-288C

 Directory of C:\Program Files (x86)\KDiff3\bin

01/30/2014  09:59 AM    <DIR>          .
01/30/2014  09:59 AM    <DIR>          ..
07/05/2012  06:29 PM               385 bin.Manifest
05/24/2004  05:46 AM            57,344 cmp.exe
05/24/2004  05:46 AM           150,528 diff.exe
05/24/2004  05:46 AM            59,392 diff3.exe
07/05/2012  09:12 AM            43,008 libgcc_s_dw2-1.dll
03/14/2008  03:21 PM         1,008,128 libiconv2.dll
05/06/2005  12:52 PM           103,424 libintl3.dll
07/05/2012  09:12 AM            11,362 mingwm10.dll
01/30/2014  09:59 AM    <DIR>          plugins
07/05/2012  05:56 PM         2,849,280 QtCore4.dll
05/06/2012  01:24 PM        10,151,424 QtGui4.dll
07/06/2012  02:11 PM               214 README-Qt.txt
07/06/2012  01:44 PM               953 README-Utilities.txt
10/24/2007  03:10 AM            79,360 regex2.dll
05/24/2004  05:46 AM            61,952 sdiff.exe
12/27/2010  10:10 AM            77,824 sed.exe
       15 File(s)     14,654,578 bytes
        3 Dir(s)  169,552,793,600 bytes free
```

```
C:\Program Files (x86)\KDiff3>dir
 Volume in drive C is OS
 Volume Serial Number is FA36-288C

 Directory of C:\Program Files (x86)\KDiff3

01/30/2014  09:59 AM    <DIR>          .
01/30/2014  09:59 AM    <DIR>          ..
01/30/2014  09:59 AM    <DIR>          bin
08/11/2012  05:27 PM            28,764 ChangeLog.txt
09/02/2011  02:13 PM            53,627 COPYING.txt
10/29/2006  12:38 PM             1,447 DIFF-EXT-LICENSE.txt
08/04/2012  07:14 PM           166,400 diff_ext_for_kdiff3.dll
08/04/2012  07:15 PM           176,640 diff_ext_for_kdiff3_64.dll
01/30/2014  09:59 AM    <DIR>          doc
08/11/2012  05:23 PM         1,060,864 kdiff3.exe
07/05/2012  06:27 PM               343 kdiff3.exe.manifest
07/05/2012  08:09 PM                25 qt.conf
08/04/2012  07:22 PM             6,664 README_WIN.txt
01/30/2014  09:59 AM    <DIR>          translations
01/30/2014  09:59 AM           219,942 Uninstall.exe
              10 File(s)      1,714,716 bytes
               5 Dir(s)  169,543,053,312 bytes free
```

WINDOWS Git Install

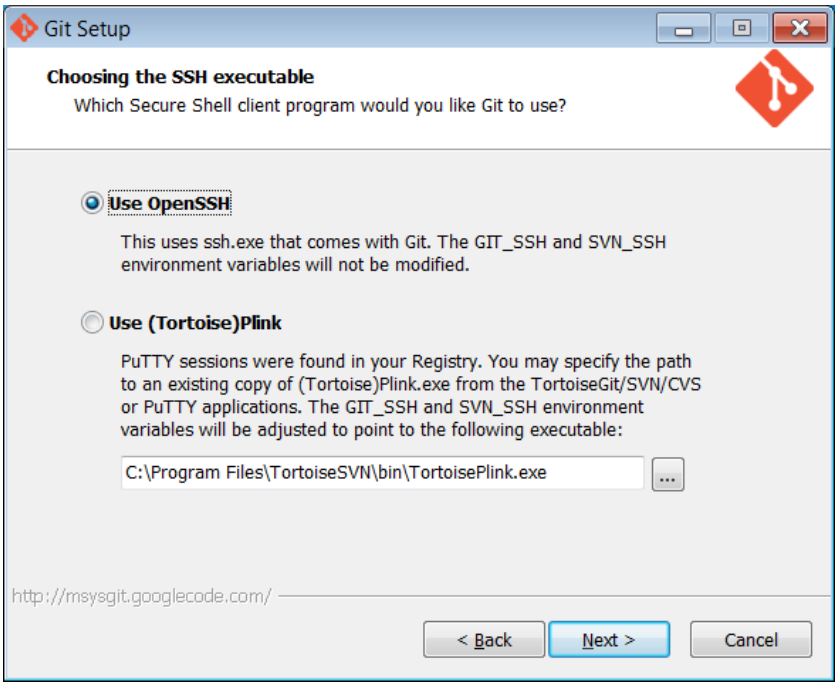
http://git-scm.com/download/win

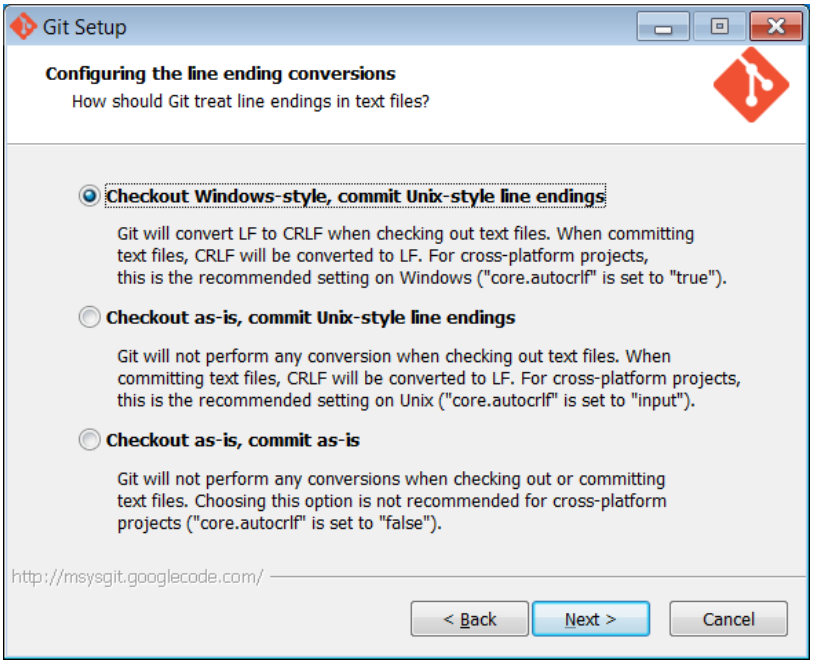


Open up cmd.exe window (notice I have BASE Cygwin, with SSH installed)

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\mydocs>ssh
usage: ssh [-1246AaCfgKkMNnqsTtVvXxYy] [-b bind_address] [-c cipher_spec]
           [-D [bind_address:]port] [-e escape_char] [-F configfile]
           [-I pkcs11] [-i identity_file]
           [-L [bind_address:]port:host:hostport]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-R [bind_address:]port:host:hostport] [-S ctl_path]
           [-W host:port] [-w local_tun[:remote_tun]]
           [user@]hostname [command]

C:\mydocs>which ssh
ssh:
       c:\cygwin\bin\ssh.EXE

C:\mydocs>
```

Close cmd.exe window





Open up NEW cmd.exe window

```
C:\mydocs>git --version
git version 1.8.5.2.msysgit.0

C:\mydocs\Solution\ThrowAwaySolution>which git
git:
    C:\Program Files (x86)\Git\cmd\git.EXE

C:\mydocs\Solution\ThrowAwaySolution>which runemacs
runemacs:
    C:\emacs\emacs-24.3\bin\runemacs.EXE
```

```
C:\mydocs>git config --global user.name "Francis Trujillo"

C:\mydocs>git config --global user.email ftrujillo@micron.com

C:\mydocs>git config --global core.editor C:\emacs\emacs-24.3\bin\runemacs.EXE

C:\mydocs>git config --global merge.tool kdiff3

C:\mydocs>git config --global diff.tool kdiff3

C:\mydocs>git config --global difftool.kdiff3.cmd "kdiff3 $LOCAL $REMOTE"

C:\mydocs>git config --global difftool.prompt false

C:\Program Files (x86)\KDiff3>git config --list
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsizelimit=2g
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=Francis Trujillo
user.email=ftrujillo@micron.com
core.editor=C:\emacs\emacs-24.3\bin\runemacs.EXE
merge.tool=kdiff3
diff.tool=kdiff3
difftool.kdiff3.cmd=kdiff3 $LOCAL $REMOTE
difftool.prompt=false
```

SSH KEYGEN on Windows (mydocs is symbolic link to my documents folder)

Setup HOME variable for ssh config file.

```
C:\mydocs\Solution\ThrowAwaySolution>echo %HOME%
C:\Users\ftrujillo\Documents

C:\mydocs\.ssh>ssh-keygen -b 1024 -t rsa -f c:/mydocs/.ssh/micron_git_rsa
Generating public/private rsa key pair.
cygwin warning:
  MS-DOS style path detected: c:/mydocs/.ssh/micron_git_rsa
  Preferred POSIX equivalent is: /cygdrive/c/mydocs/.ssh/micron_git_rsa
  CYGWIN environment variable option "nodosfilewarning" turns off this warning.
  Consult the user's guide for more details about POSIX paths:
    http://cygwin.com/cygwin-ug-net/using.html#using-pathnames
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in c:/mydocs/.ssh/micron_git_rsa.
Your public key has been saved in c:/mydocs/.ssh/micron_git_rsa.pub.
The key fingerprint is:
8a:da:40:ed:ad:0d:af:96:21:20:ff:5e:89:f0:e2:83 ftrujillo@FTRUJILLO-LAP

C:\mydocs\.ssh>cat micron_git_rsa.pub >> authorized_keys
```

Create a config file for ssh to use the proper private rsa identity file.

```
C:\mydocs\.ssh>cat config
Host fmnxvgit01.micron.com
    HostName fmnxvgit01.micron.com
    User git
    Port 22
    Hostname fmnxvgit01.micron.com
    IdentityFile ~/.ssh/micron_git_rsa
    TCPKeepAlive yes
    IdentitiesOnly yes

Host fmnxvgit01
    HostName fmnxvgit01.micron.com
    User git
    Port 22
    Hostname fmnxvgit01.micron.com
    IdentityFile ~/.ssh/micron_git_rsa
    TCPKeepAlive yes
    IdentitiesOnly yes


C:\mydocs\.ssh>dir
 Volume in drive C is OS
 Volume Serial Number is FA36-288C

 Directory of C:\mydocs\.ssh

01/31/2014  09:57 AM    <DIR>          .
01/31/2014  09:57 AM    <DIR>          ..
01/31/2014  09:37 AM               237 authorized_keys
01/31/2014  09:54 AM               211 config
01/31/2014  09:22 AM             1,242 known_hosts
01/31/2014  09:26 AM               887 micron_git_rsa
01/31/2014  09:26 AM               237 micron_git_rsa.pub
```

METASTORM Git example

========= Use the Metastorm GUI to create a Solution, SAVE =========

```
C:\mydocs\Solution>cd ThrowAway*

C:\mydocs\Solution\ThrowAwaySolution>dir
Directory of C:\mydocs\Solution\ThrowAwaySolution

01/30/2014  03:19 PM    <DIR>          .
01/30/2014  03:19 PM    <DIR>          ..
01/30/2014  03:19 PM    <DIR>          Project1
01/30/2014  03:19 PM             2,696 ThrowAwaySolution.Solution
```

Create a LOCAL git repo – This will keep the data in ./.git

```
C:\mydocs\Solution\ThrowAwaySolution>git init
Initialized empty Git repository in C:/mydocs/Solution/ThrowAwaySolution/.git/

C:\mydocs\Solution\ThrowAwaySolution>git add .

C:\mydocs\Solution\ThrowAwaySolution>git commit -m "Initial commit"
[master (root-commit) 334f270] Initial commit
 6 files changed, 3566 insertions(+)
 create mode 100644 Project1/Connections.Connection
 create mode 100644 Project1/Form1.Form
 create mode 100644 Project1/Process1.Process
 create mode 100644 Project1/Project1.BPMProj
 create mode 100644 Project1/Role.Role
 create mode 100644 ThrowAwaySolution.Solution

C:\mydocs\Solution\ThrowAwaySolution>git status
On branch master
nothing to commit, working directory clean
```

========= Use the Metastorm GUI to change Project1 to FJTProject1, SAVE =========

```
C:\mydocs\Solution\ThrowAwaySolution>dir
Directory of C:\mydocs\Solution\ThrowAwaySolution

01/30/2014  03:22 PM    <DIR>          .
01/30/2014  03:22 PM    <DIR>          ..
01/30/2014  03:22 PM    <DIR>          FJTProject1
01/30/2014  03:22 PM             2,705 ThrowAwaySolution.Solution
```

Git status is smart enough to know what was deleted and what was added. It allows you To accept the changes or discard, THEN commit.

```
C:\mydocs\Solution\ThrowAwaySolution>git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    Project1/Connections.Connection
        deleted:    Project1/Form1.Form
        deleted:    Project1/Process1.Process
        deleted:    Project1/Project1.BPMProj
        deleted:    Project1/Role.Role
        modified:   ThrowAwaySolution.Solution
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        FJTProject1/

no changes added to commit (use "git add" and/or "git commit -a")
```

```
C:\mydocs\Solution\ThrowAwaySolution>git rm -r Project1
rm 'Project1/Connections.Connection'
rm 'Project1/Form1.Form'
rm 'Project1/Process1.Process'
rm 'Project1/Project1.BPMProj'
rm 'Project1/Role.Role'
```

```
C:\mydocs\Solution\ThrowAwaySolution>git add FJTProject1
```

```
C:\mydocs\Solution\ThrowAwaySolution>git commit -m "First checkin with changes project name"
[master b972a74] First checkin with changes project name
 5 files changed, 3 insertions(+), 3 deletions(-)
 rename {Project1 => FJTProject1}/Connections.Connection (100%)
 rename Project1/Project1.BPMProj => FJTProject1/FJTProject1.BPMProj (96%)
 rename {Project1 => FJTProject1}/Form1.Form (100%)
 rename {Project1 => FJTProject1}/Process1.Process (100%)
 rename {Project1 => FJTProject1}/Role.Role (100%)
```

I forgot to add the solution file. Git status reminded me. Could've done => git add . on last add on last add above.

```
C:\mydocs\Solution\ThrowAwaySolution>git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   ThrowAwaySolution.Solution

no changes added to commit (use "git add" and/or "git commit -a")

C:\mydocs\Solution\ThrowAwaySolution>git add .

C:\mydocs\Solution\ThrowAwaySolution>git commit -m "checkin solution"
[master 1540276] checkin solution
 1 file changed, 2 insertions(+), 2 deletions(-)

C:\mydocs\Solution\ThrowAwaySolution>git status
On branch master
nothing to commit, working directory clean
```

Now, you are ready to push to a remote Git server if you have one setup already.

Gitorious server for Micron. RWILLIAMS is your contact.

Add your key here. Create a project and repo while you are there.
https://fmnxvgit01.micron.com/

```
C:\mydocs\Solution\ThrowAwaySolution>git remote add origin
git@fmnxvgit01.micron.com:metastorm/throwawaysolution.git

C:\mydocs\Solution\ThrowAwaySolution>git config --list --local
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
core.hidedotfiles=dotGitOnly
remote.origin.url=git@fmnxvgit01.micron.com:metastorm/throwawaysolution.git
remote.origin.fetch=+refs/heads/*:refs/remotes/origin/*
```

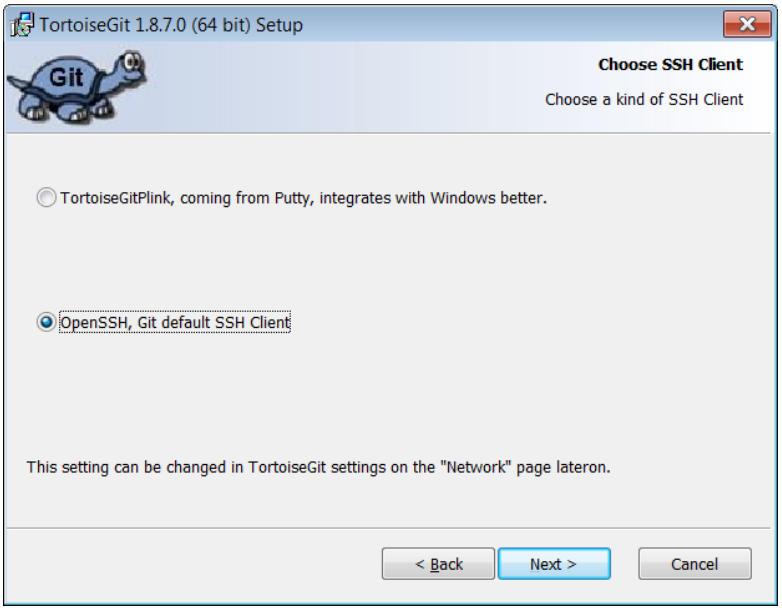
```
C:\mydocs\Solution\ThrowAwaySolution>git add .

C:\mydocs\Solution\ThrowAwaySolution>git status
On branch master
Your branch is up-to-date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   README.txt


C:\mydocs\Solution\ThrowAwaySolution>git commit -m "added a file"
[master c04627c] added a file
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt


C:\mydocs\Solution\ThrowAwaySolution>git push origin -u master
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 351 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: => Syncing Gitorious... [OK]
To git@fmnxvgit01.micron.com:metastorm/throwawaysolution.git
   1540276..c04627c  master -> master
Branch master set up to track remote branch master from origin.
```

Tortoisegit

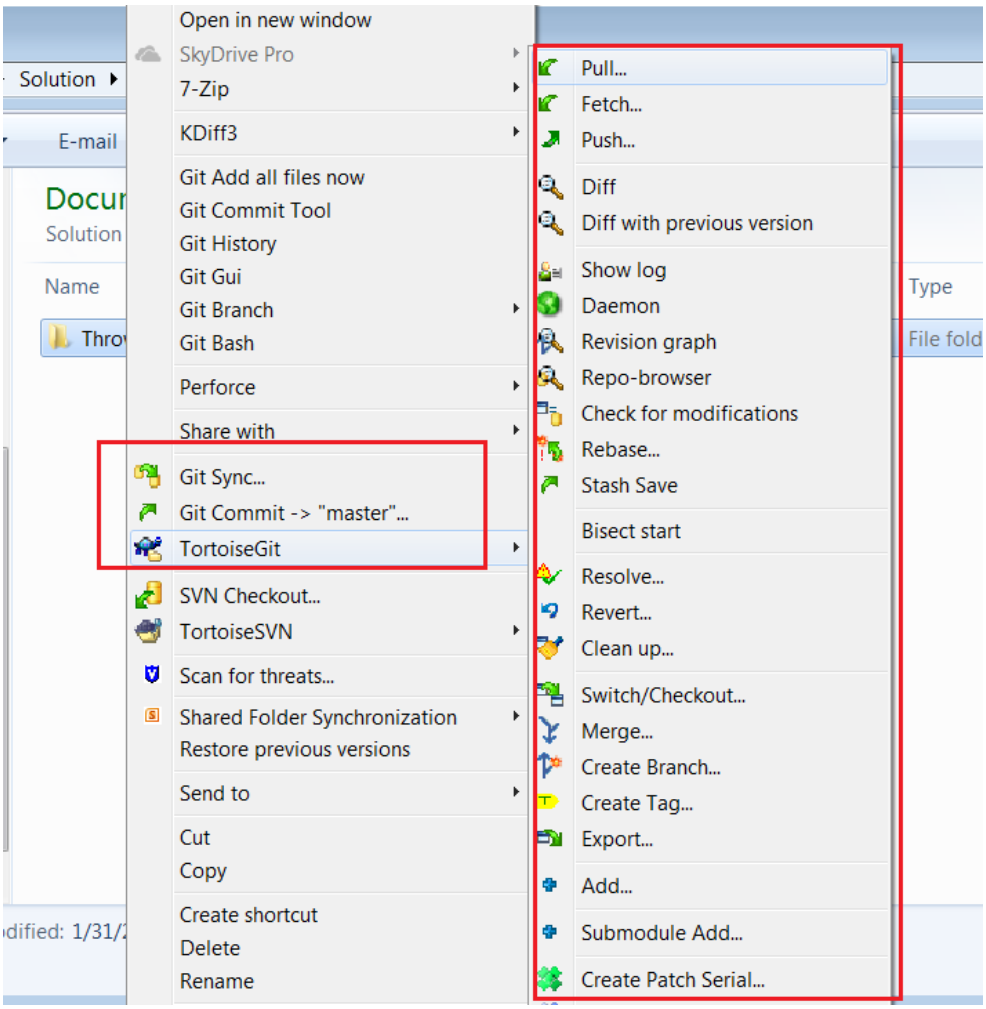
http://code.google.com/p/tortoisegit/wiki/Download

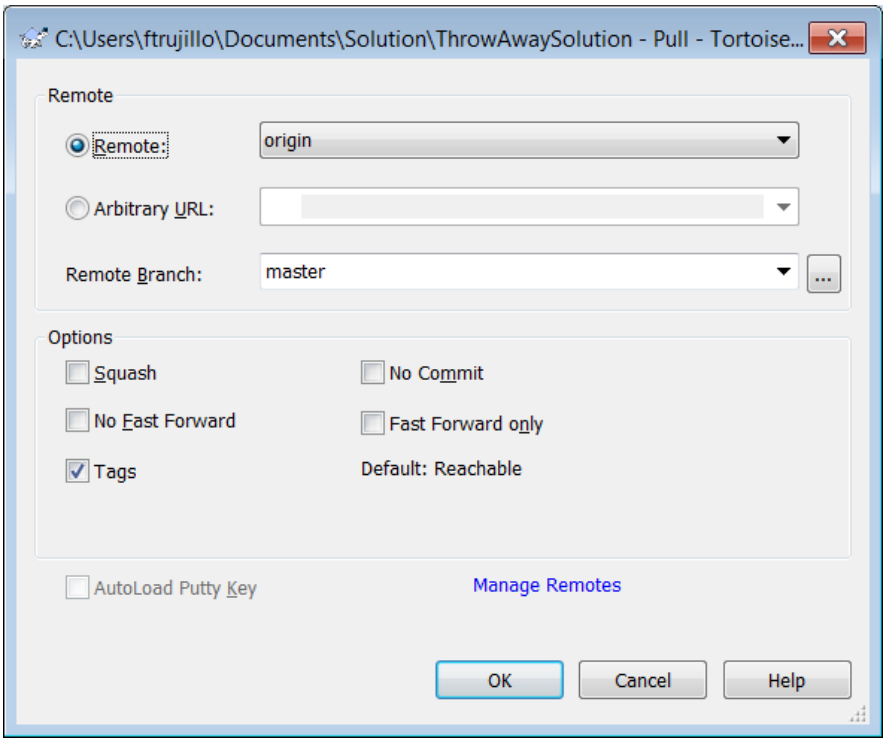
http://download.tortoisegit.org/tgit/1.8.7.0/TortoiseGit-1.8.7.0-32bit.msi

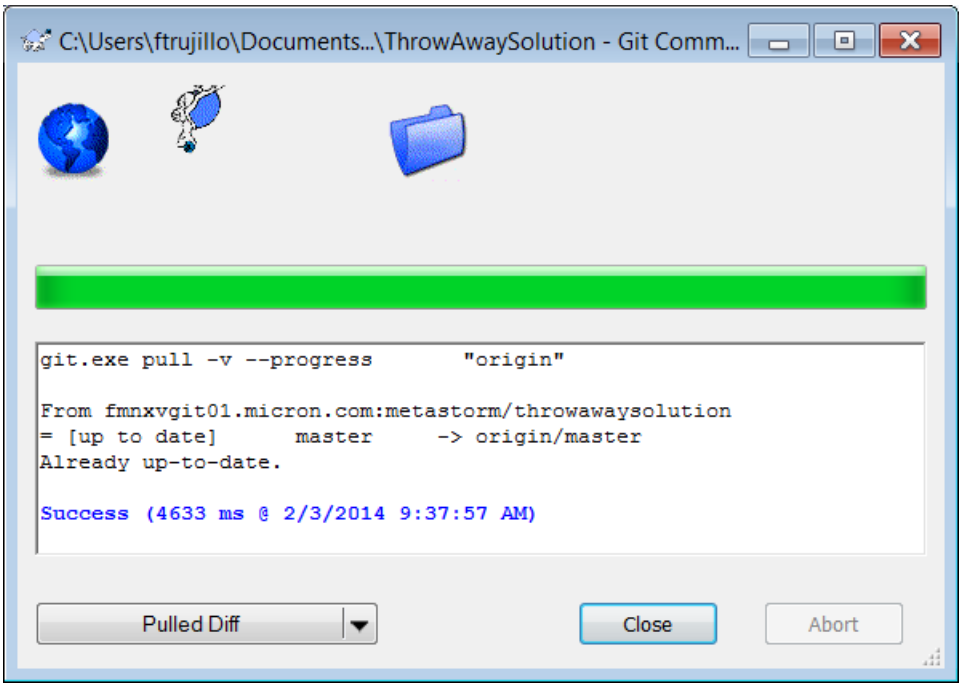
http://download.tortoisegit.org/tgit/1.8.7.0/TortoiseGit-1.8.7.0-64bit.msi

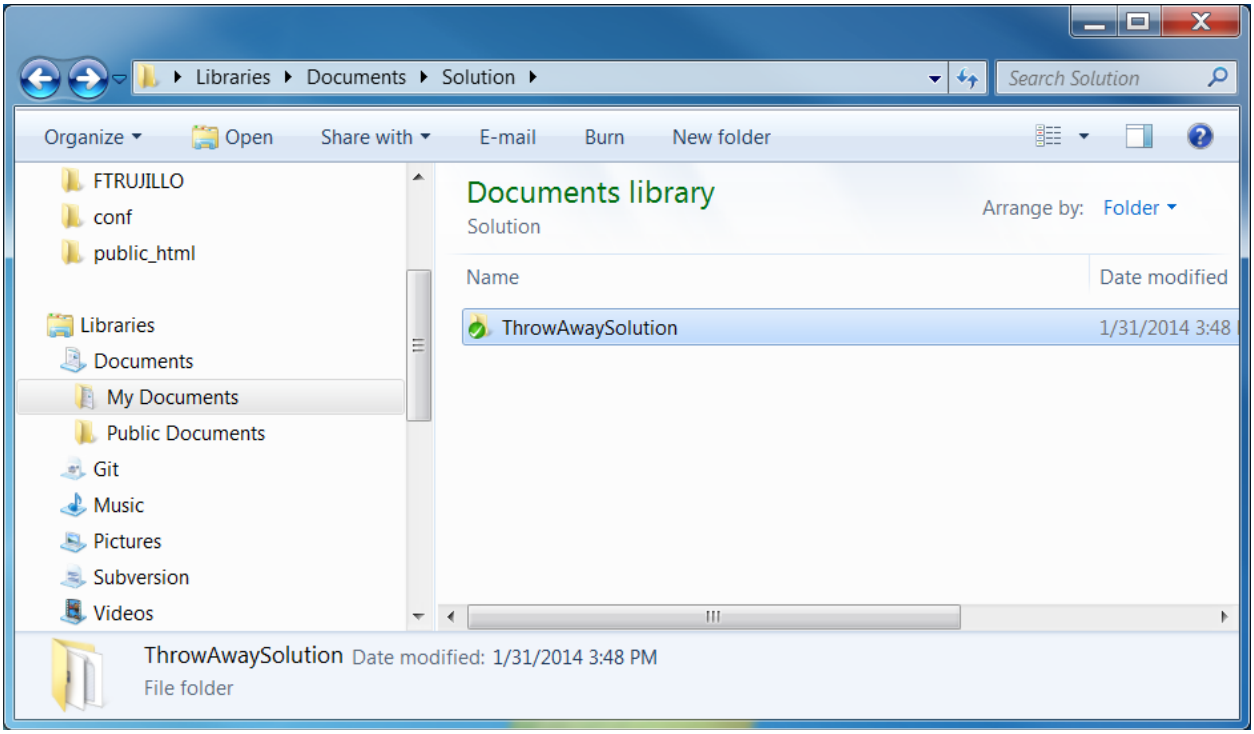


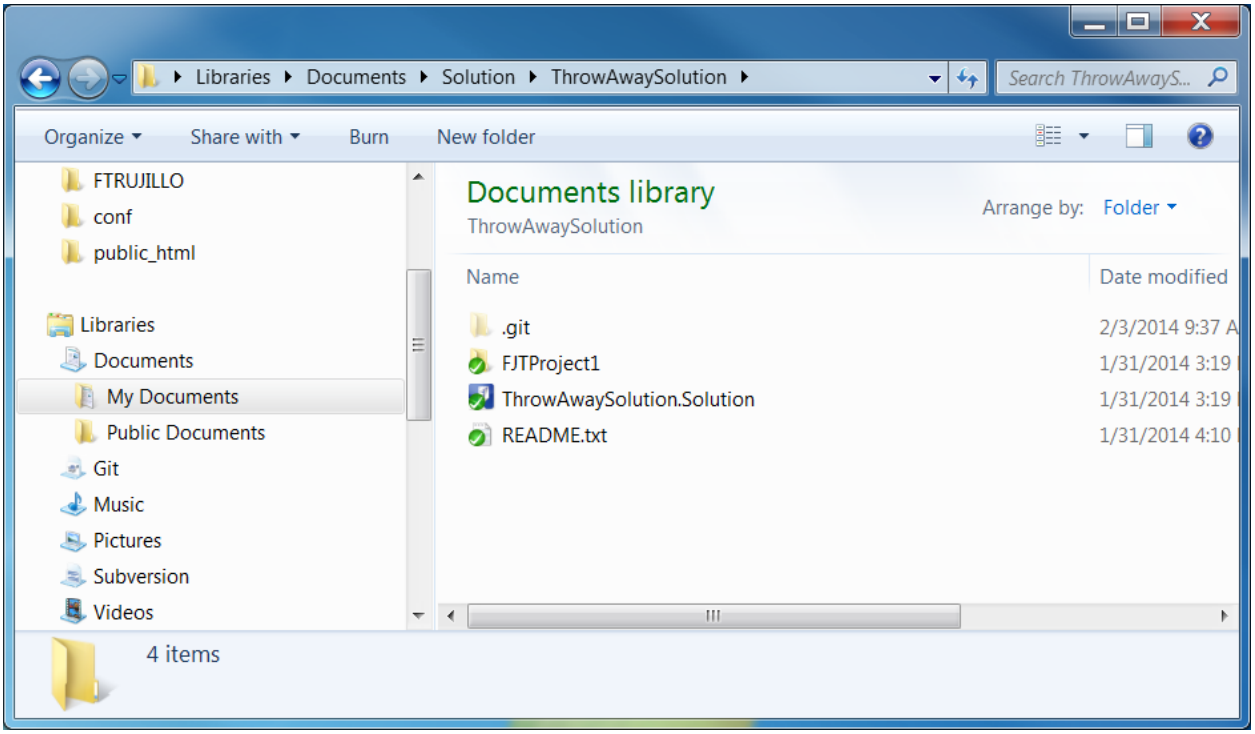
LOGOUT and back in to have TortoiseGit graphics (green checkmark, red x) to be in WinExplorer.


Open in new window
SkyDrive Pro
7-Zip
KDiff3
Git Add all files now
Git Commit Tool
Git History
Git Gui
Git Branch
Git Bash
Perforce
Share with
Git Sync...
Git Commit -> "master"...
TortoiseGit
SVN Checkout...
TortoiseSVN
Scan for threats...
Shared Folder Synchronization
Restore previous versions
Send to
Cut
Copy
Create shortcut
Delete
Rename
Pull...
Fetch...
Push...
Diff
Diff with previous version
Show log
Daemon
Revision graph
Repo-browser
Check for modifications
Rebase...
Stash Save
Bisect start
Resolve...
Revert...
Clean up...
Switch/Checkout...
Merge...
Create Branch...
Create Tag...
Export...
Add...
Submodule Add...
Create Patch Serial...
Solution
E-mail
Name
Type
File fold


C:\Users\ftrujillo\Documents\Solution\ThrowAwaySolution - Pull - Tortoise...
Remote
Remote:
origin
Arbitrary URL:
Remote Branch:
master
Options
Squash
No Commit
No Fast Forward
Fast Forward only
Tags
Default: Reachable
AutoLoad Putty Key
Manage Remotes
OK
Cancel
Help


C:\Users\ftrujillo\Documents...\ThrowAwaySolution - Git Comm...
git.exe pull -v --progress "origin"
From fmnxvgit01.micron.com:metastorm/throwawaysolution
= [up to date] master -> origin/master
Already up-to-date.
Success (4633 ms @ 2/3/2014 9:37:57 AM)
Pulled Diff
Close
Abort

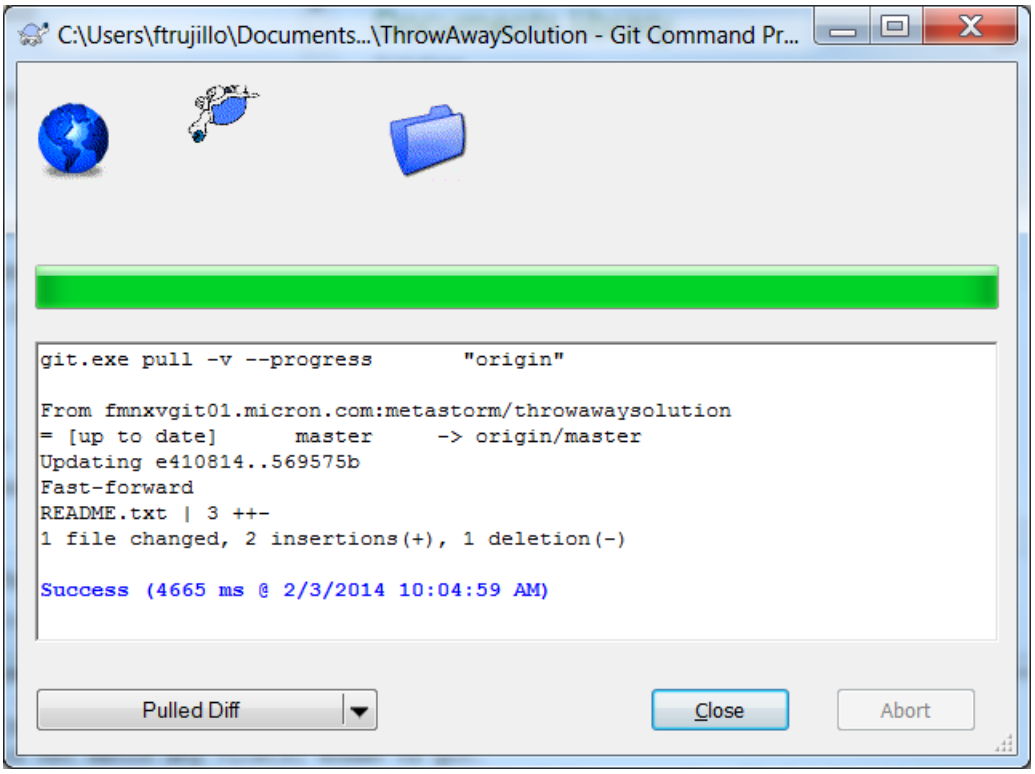

Libraries ▸ Documents ▸ Solution ▸
Search Solution
Organize
Open
Share with
E-mail
Burn
New folder
FTRUJILLO
conf
public_html
Libraries
Documents
My Documents
Public Documents
Git
Music
Pictures
Subversion
Videos
Documents library
Solution
Arrange by: Folder
Name
Date modified
ThrowAwaySolution
1/31/2014 3:48
ThrowAwaySolution Date modified: 1/31/2014 3:48 PM
File folder



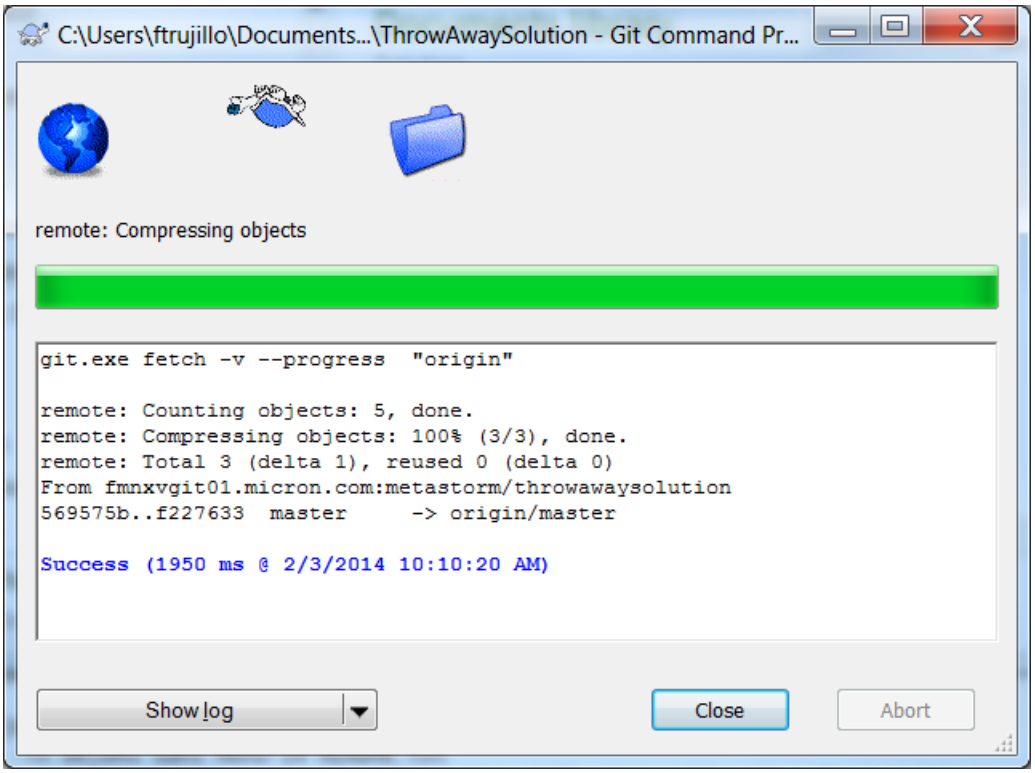
Updated a file from the linux side to show a change from another workspace.

```
nsglnxdev1.micron.com:/home/ftrujillo/WINDOWS/throwawaysolution [603]$ e README.txt &
[1] 27514
nsglnxdev1.micron.com:/home/ftrujillo/WINDOWS/throwawaysolution [604]$ git add .
[1]  + Done                          emacs -geometry 115x42+0+29 -fn DejaVu Sans Mono-14 README.txt
nsglnxdev1.micron.com:/home/ftrujillo/WINDOWS/throwawaysolution [605]$ git commit -e "Added another
comment to see push pull fetch"
error: pathspec 'Added another comment to see push pull fetch' did not match any file(s) known to
git.
nsglnxdev1.micron.com:/home/ftrujillo/WINDOWS/throwawaysolution [606]$ git commit -m "Added another
comment to see push pull fetch"
[master 569575b] Added another comment to see push pull fetch
 1 files changed, 2 insertions(+), 1 deletions(-)
nsglnxdev1.micron.com:/home/ftrujillo/WINDOWS/throwawaysolution [607]$ git push origin -u master
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 461 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
remote: => Syncing Gitorious... [OK]
To git@fmnxvgit01.micron.com:metastorm/throwawaysolution.git
   e410814..569575b  master -> master
Branch master set up to track remote branch master from origin.
```

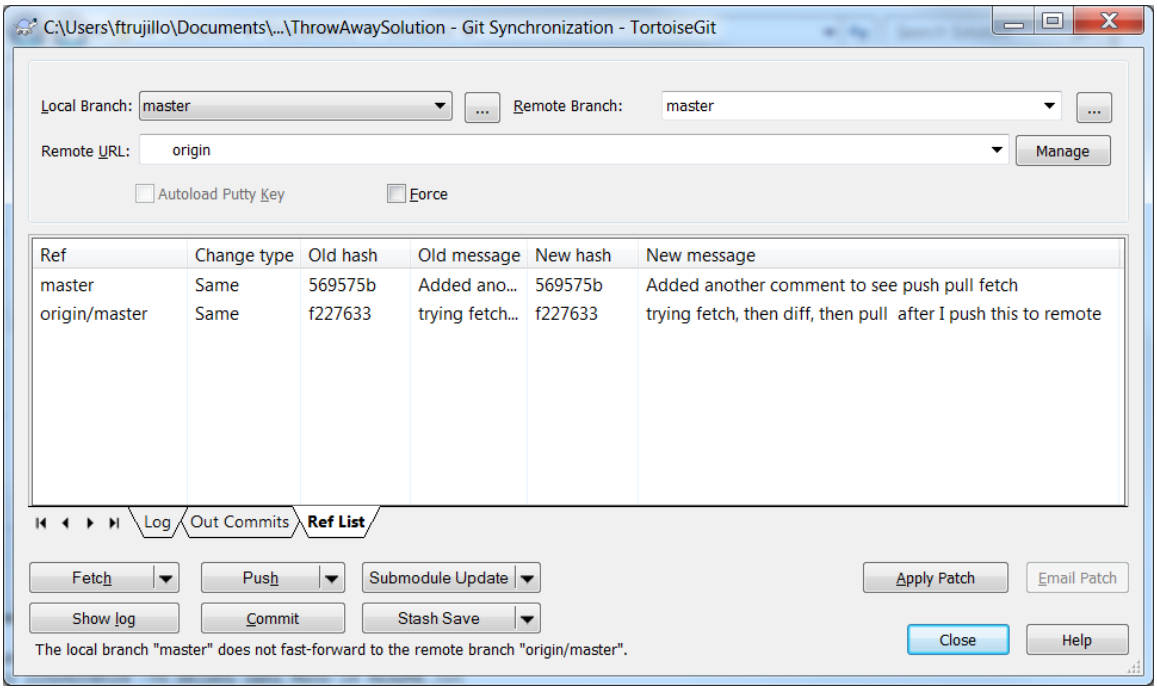
Pull if you want Changes merged into current code base. (pull == fetch + merge)

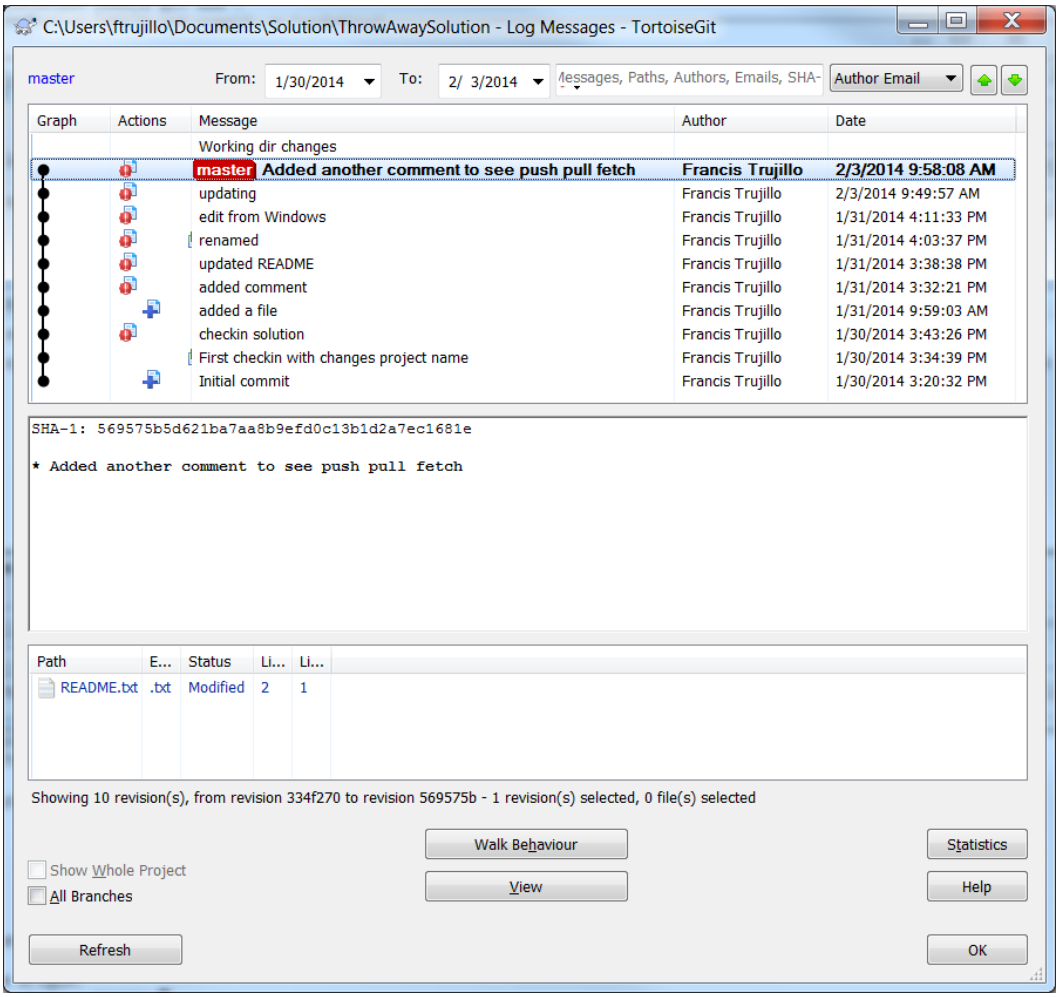


Fetch will just bring the remote changes into your local HEAD and allow you to diff.



You will need to do a PULL if you want to merge this change into working copy.
This is showing the Git Sync (fetch) The above one was a straight git fetch.
There are options to do pull, push, etc on Git Sync.





A straight git fetch shows data like this.

5

There are actually three things here: `origin master` is two separate things, and `origin/master` is one thing. Three things total.

Two branches:

- `master` is a local branch
- `origin/master` is a remote branch (which is a *local copy* of the branch named "master" on the remote named "origin")

One remote:

- `origin` is a remote

Example: pull in two steps

Since `origin/master` is a branch, you can merge it. Here's a pull in two steps:

Step one, fetch `master` from the remote `origin`. The `master` branch on `origin` will be fetched and the local copy will be named `origin/master`.

```
git fetch origin master
```

Then you merge `origin/master` into `master`.

```
git merge origin/master
```

Then you can push your new changes in `master` back to `origin`:

```
git push origin master
```

More examples

You can fetch multiple branches by name...

```
git fetch origin master stable oldstable
```

You can merge multiple branches...

```
git merge origin/master hotfix-2275 hotfix-2276 hotfix-2290
```

http://git-scm.com/docs/gitrevisions

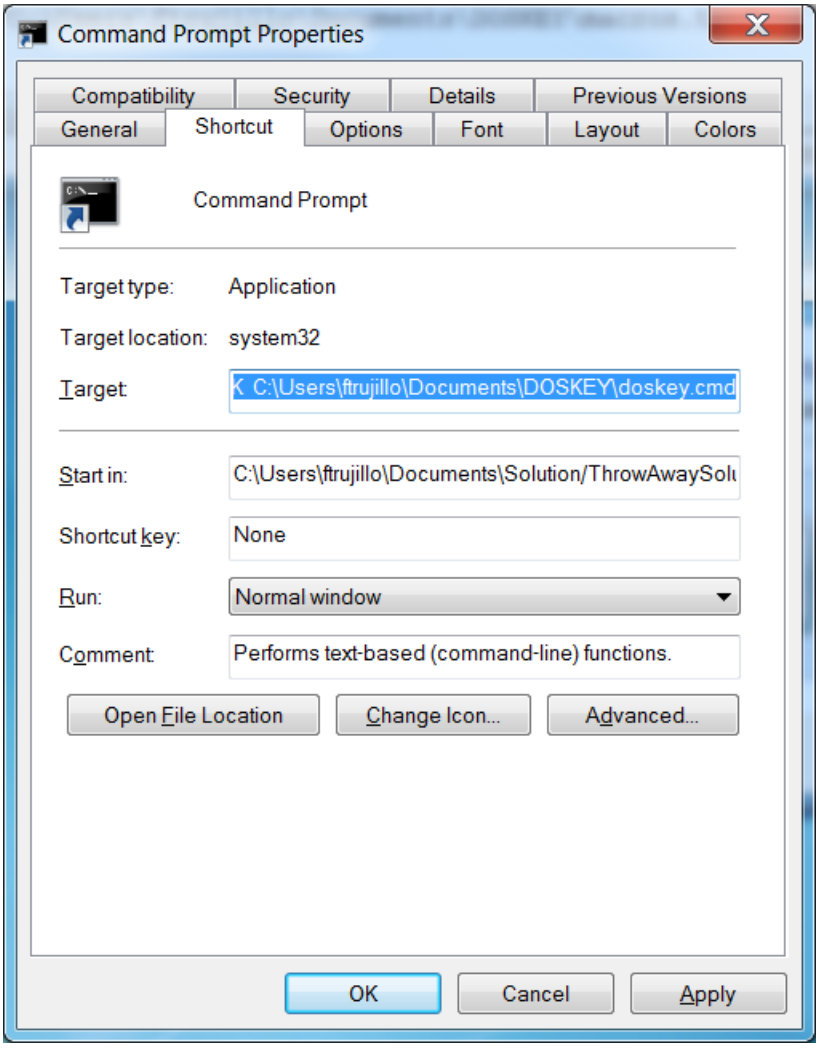
- `HEAD` names the commit on which you based the changes in the working tree.
- `FETCH_HEAD` records the branch which you fetched from a remote repository with your last git fetch invocation.
- `ORIG_HEAD` is created by commands that move your `HEAD` in a drastic way, to record the position of the `HEAD` before their operation, so that you can easily change the tip of the branch back to the state before you ran them.
- `MERGE_HEAD` records the commit(s) which you are merging into your branch when you run git merge.
- `CHERRY_PICK_HEAD` records the commit which you are cherry-picking when you run git cherry-pick.

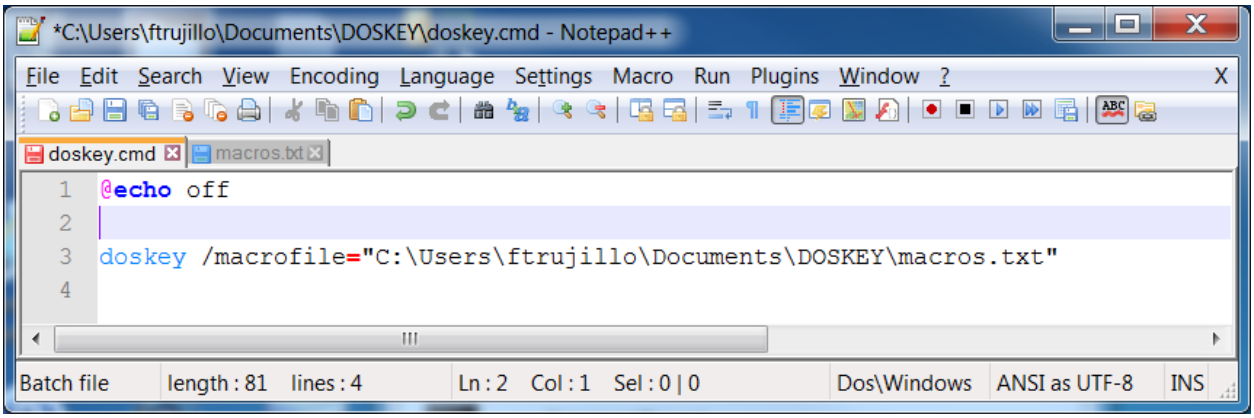
http://devblog.point2.com/2010/05/14/setup-persistent-aliases-macros-in-windows-command-prompt-cmd-exe-using-doskey/

http://ponderingdeveloper.com/2013/05/06/237/

Create a shortcut for cmd.exe and modify properties.to add reading in macros from cmd file.

TARGET => %windir%\system32\cmd.exe /K C:\Users\ftrujillo\Documents\DOSKEY\doskey.cmd





macros.txt - Windows

```
g=echo off $t echo STATUS $t echo ======== $t git status $t echo. $t git status --porcelain $t echo
on
gg=git remote update $t echo off $t echo STATUS $t echo ======== $t git status $t echo. $t echo LOG
$t echo ======== $t git --no-pager log -5 --pretty=format:"%h | %d | %an | %cn | %cd | %cr | %s" $t
echo. $t echo. $t echo BRANCHES $t echo ======== $t git branch -v $t echo. $t echo BRANCHES NOT
MERGED $t echo ==================== $t git branch --no-merged $t echo. $t echo BRANCHES ALREADY
MERGED $t echo ====================== $t git branch --merged $t echo on
ggg=type C:\Users\ftrujillo\Documents\DOSKEY\macros.txt
gggg=echo off $t echo GLOBAL $t echo. $t git config --list --global $t echo. $t echo LOCAL $t git
config --list --local $t echo on

gf=echo off $t echo FETCH $t echo. $t git fetch origin master $t echo. $t echo FETCH_TAGS $t echo.
$t git fetch origin master --tags $t echo. $t echo DIFF $t echo. $t git diff master origin/master -
-name-status $t echo on
gfetch=echo off $t echo FETCH $t echo. $t git fetch $1 $2 $t echo FETCH_TAGS $t echo. $t git fetch
--tags $1 $2 $t echo on

gd=git diff master origin/master --name-status
gdiff=git diff $1 $2 --name-status
gdt=git difftool master origin/master -- $1
gdifftool=git difftool $1 $2 -- $3

gm=git merge FETCH_HEAD
gmerge=git merge $1
gmt=git mergetool --no-prompt $1


gpull=echo off $t git pull origin master $t git pull origin master --tags $t echo on
gpush=echo off $t git push origin master $t git push origin master --tags $t echo on


gadd=echo off $t git add $1 $t git status $t echo on
grm=echo off $t git rm $1 $t git status $t echo on
gmv=echo off $t git mv $1 $2 $t git status $t echo on
gcommit=git commit -m $*
gabandon=git checkout HEAD -- $1
gunstage=git reset HEAD -- $1

glog=git log --graph --pretty=format:"%Cred%h%Creset -%C(yellow)%d%Creset %s %Cgreen(%cr) %C(bold
blue)<%an>%Creset" --abbrev-commit --date=relative

gs=git show $1
gl=git log $1
gt=git tag -a $1 $2
ga=git annotate $1

gb=git branch -v
gco=git checkout master
gcotags=git checkout tags/$1
gcobranch=git checkout $1
gnewbranch=git checkout -b $1
gdelbranch=git checkout -d $1
gforcedelbranch=git checkout -D $1

gremadd=git remote add $1 $2
gremrm=git remote remove $1
gremmv=git remote rename $1 $2

ginit=echo off $t git init . $t echo Initial git checkin > README.txt $t git add . $t git commit -m
"Initial commit" $t echo on
```

.aliases – Linux

```
alias g                'echo "STATUS\n========" ; git status ; echo ""; git status --porcelain '
alias gg               'git remote update ; echo "STATUS\n========" ; git status ; echo
"\nLOG\n========" ; git --no-pager log -5 --pretty=format:"%h | %d | %an | %cn | %cd | %cr | %s" ;
echo "\n\nBRANCHES\n========" ; git branch -v ; echo "\nBRANCHES NOT MERGED\n===================" ;
git branch --no-merged ; echo "\nBRANCHES ALREADY MERGED\n=====================" ; git branch --
merged'
alias ggg              'egrep "alias g[a-z]*" $HOME/.aliases | grep "git" | perl -pi -e "s/alias
//g;"'
alias gggg             'echo "GLOBAL + LOCAL\n" ; git config --list'

alias gf               'echo "FETCH\n" ; git fetch origin master ; echo "\nFETCH_TAGS\n" ; git fetch
origin master --tags ; echo "\nDIFF\n" ; git diff master origin/master --name-status'
alias gfetch           'echo "FETCH\n" ; git fetch \!:1 \!:2 ; echo "FETCH_TAGS\n" ; git fetch --
tags \!:1 \!:2'

alias gd               'git diff master origin/master --name-status'
alias gdiff            'git diff \!:1 \!:2 --name-status'
alias gdt              'git difftool master origin/master -- \!:1'
alias gdifftool        'git difftool \!:1 \!:2 -- \!:3'

alias gm               'git merge FETCH_HEAD'
alias gmerge           'git merge \!:1'
alias gmt              'git mergetool --no-prompt \!:1'

alias gpull            'git pull origin master ; git pull origin master --tags'
alias gpush            'git push origin master ; git push origin master --tags'

alias gadd             'git add \!:1 ; git status'
alias grm              'git rm \!:1 ; git status'
alias gmv              'git mv \!:1 \!:2 ; git status'
alias gcommit          'git commit -m \!*'
alias gabandon         'git checkout HEAD -- \!:1'
alias gunstage         'git reset HEAD -- \!:1'

alias glog             'git log --graph --pretty=format:"%Cred%h%Creset -%C(yellow)%d%Creset %s
%Cgreen(%cr) %C(bold blue)<%an>%Creset" --abbrev-commit --date=relative'

alias gs               'git show \!:1'
alias gl               'git log \!:1'
alias gt               'git tag -a \!:1 \!:2'
alias ga               'git annotate \!:1'

alias gb               'git branch -v'
alias gco              'git checkout master'
alias gcotags          'git checkout tags/\!:1'
alias gcobranch        'git checkout \!:1'
alias gnewbranch       'git checkout -b \!:1'
alias gdelbranch       'git checkout -d \!:1'
alias gforcedelbranch 'git checkout -D \!:1'

alias gremadd          'git remote add \!:1 \!:2'
alias gremrm           'git remote remove \!:1'
alias gremmv           'git remote rename \!:1 \!:2'

alias ginit            'git init . ; echo "Initial git checkin" >> README.txt ; git add . ; git
commit -m "Initial commit"'
```