

Tutorial on Version Control and Git

AMath 590, Autumn 2013

Randy LeVeque

University of Washington

Based on slides from a tutorial at the

ICERM Workshop on
Reproducibility in Computational and
Experimental Mathematics

<http://icerm.brown.edu/tw12-5-rcem>



Content is licensed under a Creative Commons Attribution 3.0 License

Version Control / Git Tutorial

- Introduction to [version control systems](#) (VCS).
- Basic Git commands
- Useful even for working alone:
 - Track changes to documents or programs,
 - Easily revert to earlier versions,
 - Determine where bugs were introduced in code,
 - Archive version of code used to obtain important results.

Version control systems

Originally developed for large software projects with many developers.

Also useful for single user, e.g. to:

- Keep track of history and changes to files,
- Be able to revert to previous versions,
- Keep many different versions of code well organized,
- Easily archive exactly the version used for results in publications,
- Keep work in sync on multiple computers.

Server-client model:

Original style, still widely used (e.g. CVS, Subversion)

One **central repository** on server.

Developers' workflow (simplified!):

- Check out a **working copy**,
- Make changes, test and debug,
- Check in (**commit**) changes to repository (with comments).
This creates new **version number**.
- Run an **update** on working copy to bring in others' changes.

The system keeps track of **diffs** from one version to the next (and info on who made the changes, when, etc.)

A **changeset** is a collection of **diffs** from one commit.

Server-client model:

Only the server has the full history.

The working copy has:

- Latest version from repository (from last `checkout`, `commit`, or `update`)
- Your local changes that are not yet committed.

Server-client model:

Only the server has the full history.

The working copy has:

- Latest version from repository (from last `checkout`, `commit`, or `update`)
- Your local changes that are not yet committed.

Note:

- You can retrieve older versions from the server.
- Can only *commit* or *update* when connected to server.
- When you *commit*, it will be seen by anyone else who does an *update* from the repository.

Often there are `trunk` and `branches` subdirectories.

Distributed version control

Examples: Git, Mercurial, Bazaar

These use a distributed model:

When you clone a repository you get all the history too,

All clones are created equal, no “server repository”.

With git, All history stored in `.git` subdirectory of top directory.

Usually don't want to mess with this!

Distributed version control

Advantages of distributed model:

- You can commit changes, revert to earlier versions, examine history, etc. without being connected to server.
- Also without affecting anyone else's version if you're working collaboratively. Can commit often while debugging.
- No problem if server dies, every clone has full history.

Distributed version control

Advantages of distributed model:

- You can commit changes, revert to earlier versions, examine history, etc. without being connected to server.
- Also without affecting anyone else's version if you're working collaboratively. Can commit often while debugging.
- No problem if server dies, every clone has full history.

For collaboration, will need to **fetch** changes from other versions and **merge** into yours.

Distributed version control

Advantages of distributed model:

- You can commit changes, revert to earlier versions, examine history, etc. without being connected to server.
- Also without affecting anyone else's version if you're working collaboratively. Can commit often while debugging.
- No problem if server dies, every clone has full history.

For collaboration, will need to [fetch](#) changes from other versions and [merge](#) into yours.

May have server version that all developers can fetch from, and some can [push](#) to, for example hosted on [Github](#).

Using git in solo mode

The following examples will be performed during the tutorial.

Try to work through them on your own computer!

Some Git references

- <http://help.github.com/>
- <http://gitref.org/index.html>
- Git book
<http://git-scm.com/book/en/Getting-Started-Git-Basics>
- List of 10 recommended tutorials
<http://sixrevisions.com/resources/git-tutorials-beginners/>
- Github online tutorial
<http://try.github.com/>

More Git references

- Fernando Perez's blog has many useful links to get started.
<http://fperez.org/py4science/git.html>
- Git Parable gives a good intro to the concepts.
<http://tom.preston-werner.com/2009/05/19/the-git-parable.html>
- gitk introduction has a good description of merging.
<http://lostechies.com/joshuaflanagan/2010/09/03/use-gitk-to-understand-git/>
- AMath 583 class notes:
<http://faculty.washington.edu/rjl/classes/am583s2013/notes/git.html>

Using git in solo mode

1. Install git: see <http://git-scm.com/>
2. In any directory:

```
$ git init    # creates .git subdirectory
$ git add .   # track all files
$ git commit -m "initial commit"
```

This takes a snapshot of all files (recursively in subdirectories).

Can selectively add a subset of files instead, e.g.

```
$ git add README.txt *.tex
```

Important Git concepts

Working directory: The current state of your files.

Index: List of new or modified files that will be included in next snapshot. Use `git add` to add files to index.

Commit: the command `git commit` takes a snapshot of the current state of your working directory, but only including new or modified files if they are listed in the [index](#).

Useful git commands

```
$ git help
```

```
$ git help <topic>
```

```
$ git add <filename>
```

```
$ git commit -m <message>
```

```
$ git status | more
```

```
$ git diff | more
```

```
$ git diff <filename> | more
```


Git demo

```
$ mkdir gitdemo
```

```
$ cd gitdemo
```

```
$ git init
```

```
Initialized empty Git repository  
in /Users/rjl/gitdemo/.git/
```

```
$ cat > file1.txt
```

```
First version of file.
```

```
<ctrl>-D # to end entry into file
```

```
$ ls -a
```

```
./      ../      .git/    file1.txt
```

Git status

```
$ git status

# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..."
#     to include in what will be committed)
#
#   file1.txt

nothing added to commit but untracked
files present (use "git add" to track)
```

Git status and commit

```
$ git add file1.txt
```

```
$ git status
```

```
# On branch master
```

```
#
```

```
# Initial commit
```

```
#
```

```
# Changes to be committed:
```

```
#   (use "git rm --cached <file>..." to unstage)
```

```
#
```

```
#   new file:   file1.txt
```

```
#
```

```
$ git commit -m "initial commit"
```

```
[master (root-commit) e84a77e] initial commit
```

```
1 files changed, 1 insertions(+), 0 deletions(-)
```

```
create mode 100644 file1.txt
```

Git commit and log

```
$ git commit -m "initial commit"
[master (root-commit) e84a77e] initial commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 file1.txt
```

```
$ git log
commit e84a77e8afb9c83c37000e698795c350a931b512
Author: Randy LeVeque <rjl@uw.edu>
Date:   Wed Jul 13 08:51:55 2011 -0700
```

```
    initial commit
```

The commit has a name which is a 40-digit hexadecimal string.

SHA-1 hash code based on all contents and modification times of files.

Git branch

```
$ git branch  
* master
```

There is only one branch named **master**.

Git diff

```
$ # edit the file...
```

```
$ cat file1.txt  
Second version of file.  
Added a second line.
```

```
$ git diff  
diff --git a/file1.txt b/file1.txt  
index 2ac6b14..6ef2b34 100644  
--- a/file1.txt  
+++ b/file1.txt  
@@ -1 +1,2 @@  
-First version of file.  
+Second version of file.  
+Added a second line.
```

Git demo

```
$ git status
# On branch master
# Changes not staged for commit:
#
#   modified:   file1.txt
#
no changes added to commit
  (use "git add" and/or "git commit -a")
```

Git demo

```
$ git add file1.txt

$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   file1.txt
#
```

We have added `file1.txt` to the [index](#) of files to be committed.

Git demo

```
$ git commit -m "changed line and added second"
[master b1294af] changed line and added second
1 files changed, 2 insertions(+), 1 deletions(-)
```

```
$ git log
commit b1294aff9edb80377096fd861fa53a88af3fb1fc
Author: Randy LeVeque <rjl@uw.edu>
Date:   Wed Jul 13 09:01:57 2011 -0700
```

changed line and added second

```
commit e84a77e8afb9c83c37000e698795c350a931b512
Author: Randy LeVeque <rjl@uw.edu>
Date:   Wed Jul 13 08:51:55 2011 -0700
```

initial commit

Git demo

```
$ git log --pretty=oneline
```

```
b1294aff9edb80377096fd861fa53a88af3fb1fc changed line an  
e84a77e8afb9c83c37000e698795c350a931b512 initial commit
```

```
$ git help log # shows many more options
```

Git demo

```
$ cat > file2.txt
Another file
<ctrl>-D
```

```
$ git status
# On branch master
# Untracked files:
#
#   file2.txt
```

Note: Doesn't show file1.txt since it hasn't been modified since last commit.

file2.txt is not being tracked.

Git demo

```
$ git add file2.txt
```

```
$ git status -s
```

```
A file2.txt
```

```
$ # edit file1.txt
```

```
$ git status -s
```

```
M file1.txt
```

```
A file2.txt
```

M in column 2: Modified file has **not** been added to index.

Git demo

```
$ cat file1.txt
Third version of file.
Added a second line.
```

```
$ git diff
diff --git a/file1.txt b/file1.txt
index 6ef2b34..40afc7a 100644
--- a/file1.txt
+++ b/file1.txt
@@ -1,2 +1,2 @@
-Second version of file.
+Third version of file.
  Added a second line.
```

Git demo

```
$ git status -s  
M file1.txt  
A file2.txt
```

```
$ git add -u # add all modified files to index
```

```
$ git status -s  
M file1.txt  
A file2.txt
```

Git demo

```
$ git status -s  
M file1.txt  
A file2.txt
```

```
$ git diff # shows diff's between working copy  
$          # and index: there are none
```

Git demo

```
$ git diff HEAD # nickname for last commit
```

```
diff --git a/file1.txt b/file1.txt
```

```
index 6ef2b34..40afc7a 100644
```

```
--- a/file1.txt
```

```
+++ b/file1.txt
```

```
@@ -1,2 +1,2 @@
```

```
-Second version of file.
```

```
+Third version of file.
```

```
  Added a second line.
```

```
diff --git a/file2.txt b/file2.txt
```

```
new file mode 100644
```

```
index 0000000..b0b9fc8
```

```
--- /dev/null
```

```
+++ b/file2.txt
```

```
@@ -0,0 +1 @@
```

```
+Another file
```


Git demo

```
$ git commit -m "changed file1 and added another"  
[master 5f4e886] changed file1 and added another  
 2 files changed, 2 insertions(+), 1 deletions(-)  
 create mode 100644 file2.txt
```

```
$ mkdir testdir
```

```
$ git status  
# On branch master  
nothing to commit (working directory clean)
```

Note: Tracks only files, not directories!

Git demo

```
$ cat > testdir/file3.txt    # create a new file
In the subdirectory.
<ctrl>-D
```

```
$ git status
# On branch master
# Untracked files:
#   testdir/
```

```
$ git status -s
?? testdir/
```

```
$ git add testdir
$ git status -s
A testdir/file3.txt
```

```
$ git commit -m "testing a subdirectory"
[master 765f3a3] testing a subdirectory
```

Git demo

```
$ git commit -m "testing a subdirectory"  
[master 765f3a3] testing a subdirectory
```

```
$ git log --pretty=oneline  
765f3a3f... testing a subdirectory  
5f4e8860... changed file1 and added another  
b1294aff... changed line and added second  
e84a77e8... initial commit
```

Graphical tools, e.g. gitk

The screenshot shows the gitk graphical interface. At the top, a commit history graph displays a sequence of commits: an initial commit, followed by several changes to 'gitdemo', and a final merge of 'master' into 'new-feature'. The current commit is highlighted in green.

Below the graph, the commit details for SHA1 ID `5ce8a4c3a05d6c0b7bd8d7da45cc9684590047ab` are shown. The commit message is: "Merged master into new-feature and changed comment". The author and committer are Randy LeVeque <rjl@ned>, dated 2012-12-09 08:13:18. The parent commit is `be3c170cb66bfc8b15f9de05c3c2d25d429ad04` and the child is `e7213788dc84c7366f8221f4c81107477d995f37`. The branch is `master`.

The diff view shows the changes to `file1.txt`. The diff is as follows:

```
----- file1.txt -----
index 40afc7a..6d6422f 100644
@@ -1,2 +1,2 @@
 Third version of file.
-Added a second line.
+Added a 2nd line. # try this feature
```

Part 2

- Creating branches
- Committing to different branches
- Fetching changes, merging, etc.
- Deleting a branch

Git branches

A new repository has a single branch with named `master`.

Create a new branch:

```
$ git branch new-feature
$ git branch
* master
  new-feature
```

Switch to the branch:

```
$ git checkout new-feature
Switched to branch 'new-feature'

$ git branch
  master
* new-feature
```

Git branches

Any changes committed will now be on this branch until checking out another.

Try the following:

- Make a change on `new-feature`
- Use `git add` and `git commit` to snapshot.
- Check out master branch: `git checkout master`
- Make a different change and commit it.
- Use `gitk` to examine state of repository.

Git branches

Also note that `git log` give different things on each branch.

To see changes since the two diverged...

Commits to new-feature only:

```
$ git log master..new-feature
```

Commits to master since divergence:

```
$ git log new-feature..master
```


Git merge

To merge new changes to `master` into `new-feature`:

```
$ git checkout new-feature  
$ git merge --no-commit master
```

This merges changes into working directory.

If there are **conflicts** then these must be fixed and the files added to the index with `git commit`

Then can do

```
$ git commit -m "Merged master into new-feature"
```

Git merge

Once new feature is fully debugged and incorporates changes to `master`, merge it into `master`:

```
$ git checkout master
$ git merge new-feature
```

This merges changes into working directory.

This should be a “fast-forward” commit.

Now can delete branch:

```
$ git log master..new-feature      # make sure!
$ git branch -d new-feature
```

Part 3

- Collaboration
- Cloning a repository
- Fetching changes, merging, etc.
- Start with two clones on your own computer.

Git clone

```
$ cd ..  
$ git clone gitdemo gitdemo2  
Cloning into gitdemo2...  
done.  
  
$ cd gitdemo2  
  
$ git log --pretty=oneline  
765f3a3f... testing a subdirectory  
5f4e8860... changed file1 and added another  
b1294aff... changed line and added second  
e84a77e8... initial commit  
  
$ git remote -v  
origin  /Users/rjl/gitdemo (fetch)  
origin  /Users/rjl/gitdemo (push)
```

Adding remotes

A cloned repository will have one remote named `origin`.

You can add remotes using:

```
$ git remote add <name> <address>
```

where `<name>` is any name you want to use to refer to it.

The `<address>` can be...

- Path to another directory on your computer,
- Another computer, e.g.
`rjl@mycomputer.washington.edu:gitrepos/gitdemo`
- A web hosting site, e.g.
`git@github.com:rjleveque/gitdemo`

Fetching history from a remote repository

For example, using the remote `origin`...

```
$ git fetch origin
```

This fetches history (all commits) but does not change your version!

You can refer to branches of the remote by, e.g. `origin/master`.

Comparing to branches on a remote repository

You can do things like...

```
$ git log origin/master
```

```
$ git log master..origin/master
```

```
$ git diff master origin/master -- file1.txt
```

To merge a remote branch into working directory:

```
$ git branch # see what local branch you're on
```

```
$ git checkout ... # if you want to change
```

```
$ git merge origin/master
```

Fetching changes from origin

```
$ cd ../gitdemo
$ #edit file2.txt
$ git add -u
$ git commit -m "a commit in gitdemo"
[master bd166a5] a commit in gitdemo

$ cd ../gitdemo2
$ more file2.txt # hasn't changed!
Another file

$ git fetch origin
remote: Counting objects: 5, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From /Users/rjl/gitdemo
    765f3a3..bd166a5  master    -> origin/master
```


Git diff

```
$ git diff origin/master

diff --git a/file2.txt b/file2.txt
index 8def468..b0b9fc8 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1,2 +1 @@
  Another file
-Added a line to another file.
```

Note: Would have to delete line from
origin/master/file2.txt
to make it agree with working copy.

Git merge

```
$ git diff HEAD origin/master
diff --git a/file2.txt b/file2.txt
index b0b9fc8..8def468 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1,2 @@
  Another file
+Added a line to another file.
```

These are the changes to be made when we merge...

```
$ git merge origin/master
Updating 765f3a3..bd166a5
Fast-forward
 file2.txt |      1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

Git merge

Harder case: File has changed in different ways in two places.

git will attempt to merge changes, but cannot if they are on same line...

```
$ cd ../gitdemo
$ #edit file2.txt
$ git add -u
$ git commit -m "change in gitdemo"

$ cd ../gitdemo2
$ #make a different edit to file2.txt
$ git add -u
$ git commit -m "change in gitdemo2"
```

Git merge

```
$ git fetch origin
   bd166a5..76f0bc9  master    -> origin/master
```

```
$ git diff HEAD origin/master
diff --git a/file2.txt b/file2.txt
index 17cfc40..c10f444 100644
--- a/file2.txt
+++ b/file2.txt
@@ -1,2 +1,2 @@
   Another file
-Different change in gitdemo2
+Added a line to another file - then change it.
```

Git merge

```
$ git status -s # make sure no uncommitted changes
$
$ git merge origin/master
```

```
Auto-merging file2.txt
CONFLICT (content): Merge conflict in file2.txt
Automatic merge failed;
fix conflicts and then commit the result.
```

Must edit by hand to create proper file.

Git merge

```
$ cat file2.txt
```

```
Another file
```

```
<<<<<< HEAD
```

```
Different change in gitdemo2
```

```
=====
```

```
Added a line to another file - then change it.
```

```
>>>>>> origin/master
```

Must edit by hand to create proper file.

Then **add** to index and **commit**.

Git merge

Or... to keep local version:

```
$ git checkout HEAD file2.txt
```

```
$ cat file2.txt
```

```
Another file
```

```
Different change in gitdemo2
```

```
$ git status -s
```

```
$
```

Or... to take file as in origin:

```
$ git checkout origin/master file2.txt
```

```
$ cat file2.txt
```

```
Another file
```

```
Added a line to another file - then change it.
```

```
$ git status -s
```

```
M file2.txt
```

Part 4

- Remote repositories
- Public hosting sites, e.g. Github, Bitbucket
- Sharing via ssh

Adding remotes

A cloned repository will have one remote named `origin`.

You can add remotes using:

```
$ git remote add <name> <address>
```

where `<name>` is any name you want to use to refer to it.

The `<address>` can be...

- Path to another directory on your computer,
- Another computer, e.g.
`rjl@mycomputer.washington.edu:gitrepos/gitdemo`
- A web hosting site, e.g.
`git@github.com:rjleveque/gitdemo`

Web hosting repositories

Github: <https://github.com>

Repository for open source hosting

Many open sources projects use it, including Linux kernal.

Public repositories free.

Provides wiki, issue tracking, source browsing. See e.g.:

<https://github.com/ipython/ipython>

Bitbucket (<https://bitbucket.org/>)

and Google code (<http://code.google.com/>)

can also be used for hosting git repositories.

Create an account at...

<http://bitbucket.org/>

Then click on “Create” and follow instructions to make a new repository.

Learn about bitbucket:

<https://confluence.atlassian.com/display/BITBUCKET/Bitbucket+101>

Basic steps:

- Create account
- Create repository on bitbucket webpage
- Issue invitation to collaborators
- Set permissions if you want them to be able to push
- `git clone` repository on your machine
- Add files, make changes, commit, etc.
- `git push origin` to push history
- `git fetch origin` to fetch others' changes
- `git merge origin/master` to merge into yours
- (Fix merge issues if necessary and commit)
- `git push origin` to push new commit from merge

ssh repositories

Create a bare repository (no working directory):

```
$ git --bare init
```

Then people can clone it using:

```
$ git clone <user>@<machine>:<path>
```

if they have ssh access set up.

See for example

- <http://git-scm.com/book/ch4-1.html>
- **ssh authentication for Bitbucket**