Git staging and committing

Reuven M. Lerner, PhD
reuven@lerner.co.il • http://lerner.co.il/

Text commands

- Yes, Git uses text commands!
- You can use a GUI, and may want to do so
- But *all* Git documentation assumes that you know (and use) the text commands
- So prepare to know them, even if you don't use them! We'll use the command-line version in this course, but will discuss GUIs later on.

Adding a file

- Our repository is empty: Let's add a file to it!
- I'll use Ruby, but you can use anything.
  - Really, anything at all can be stored in Git
- Git doesn't store diffs
  - Binary files are OK, and don't get special treatment

Add a file!

- I have created a file, "hello.rb", in the repository's main directory. The contents of the file are the single line:

```
puts "Hello, world"
```

Warning!

- Never, ever create files inside of the .git directory
- That directory is *only* for Git administrative files
- Always add files *outside* of the .git directory
- Besides, files in the repository's .git directory cannot be added to the repository

Commits

- In CVS and SVN, we "commit a file"
- In Git, we don't think that way!
- A commit is a snapshot of our repository at a given point in time.
- We can choose what is in and out of a commit with "git add". This adds a file to the *commit*, not the repository
- You can (and should) use "git status" to check things

Remember:
We create commits

- One of the hardest ideas for people to get used to in Git is that you "create commits."
- "git add" is used to add files to a commit that you are creating
- "git commit" finalizes and seals the commit, adding it to Git's database
- Any change you make must be added.
- A commit may contain more than one changed file

What does Git think?

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#  (use "git add <file>..." to include in what will be committed)
#
#   hello.rb
nothing added to commit but untracked files present (use "git add" to track)
```

git status

- This is one of the most important commands you'll use
- "git status" tells you:
  - What branch your'e on
  - What files will be included in the upcoming commit
  - What files haven't yet been added to the repository?
  - What files have changed, and need to be added to the upcoming commit
  - Have you finished merging branches?

Wow!

- That's a lot to take in
- Let's consider it one piece at a time

What does Git think?

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    hello.rb
nothing added to commit but untracked files present (use "git add"
to track)
```

Branches

- We'll talk more about branches later.
- But consider this: Every commit builds on a previous one. It's most common to build on a named commit, aka a "branch"
- The default branch is “master" (sort of like the “trunk” in SVN)
- If you don't say otherwise, then you're always on "master"

What does Git think?

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#   (use "git add <file>..." to include in what will be
committed)
#
#   hello.rb
nothing added to commit but untracked files present (use "git
add" to track)
```

Initial commit

- Git knows the repository is empty
- Your first commit has yet to take place
- Often, the first commit adds a few basic files, or the skeleton of a project
- But really, this isn't a big deal

What does Git think?

```
$ git status
# On branch master
#
# Initial commit
#
# Untracked files:
#  (use "git add <file>..." to include in what will be committed)
#
#   hello.rb
nothing added to commit but untracked files present (use "git add" to track)
```

Untracked files

- Git is saying:
  - There are some files here
  - I've never seen these files before
  - If you want me to keep track of them, you'll need to tell me to notice them
  - Use "git add filename" to add a file

```
$ git add hello.rb
$
```

Notice

- There's no output from "git add"
- When we ask Git for status, we're told that there is now a single change to be committed
- The file hasn't been committed to Git
- Instead, it has been "staged," meaning that it will now be part of the commit, whenever we finish it

Staging vs. committing

- This takes a while to get used to
  - But it can be very helpful and useful
- Commits may contain any number of files
- Commits normally contain staged files
- Use the staging area to create a useful commit

Staging confusion

- It's bad enough to have to learn and work with staging in Git
- But what's worse is that three different terms are used for the same thing!
- If you see a mention of the "staging area," the "index," or the "cache," there is no difference between them

In other words

- To commit changes, you must:

```
git add FILENAME
```

- Now the file will be added to the coming commit. You create that commit with:

```
git commit –m 'Commit message' FILENAME
```

git status

- The "git status" command is your best friend
- Use it often to find out what files need to be add, were staged, and will be committed

After adding...

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

Remember

- We are creating one "commit"
- We can add any number of files to the commit
- "git add" is our way to add files
- Git will warn us if one or more files haven't been seen before ("untracked") or if they have been modified, but aren't part of the current commit

Unstaging

- Did you accidentally stage a file? That can happen, especially at the beginning
- Remove it (as Git tells you) with:

```
git reset HEAD hello.rb
```

Unstaging all

- To unstage all staged files, use “git reset HEAD”, without staging one or more filenames:

```
git reset HEAD
```

Adding multiple files

- You can add any number of files to your commit, in any combination of directories
- Use Unix wildcards for extra fun

```
git add *.rb

git add a.php b.php images/c.png

git add .    # Add the current directory!
```

Empty directories

- Git stores files, not directories
- Believe it or not, Git really doesn't deal too well with directories.
- If there are files in a directory, then Git is fine
- But you cannot commit an empty directory! Git just won't allow it.
- You can create a dummy file in a directory to ensure that Git sees it and will commit it

Now commit!

- Is your staging area ready?
  - (Did you run “git status” to be sure?)
- Commit everything in the staging area

```
git commit
```

- This will open your editor (i.e., whatever is defined as EDITOR or VISUAL), with a file that looks like…

Commit message

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#   new file:   hello.rb
#
```

Commit message

```
I added a file
```

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#    new file:   hello.rb
#
```

Commit is done

```
[master (root-commit) b53dd54] I added a file
1 files changed, 1 insertions(+), 0 deletions(-)
create mode 100644 hello.rb
```

What is this?

```
[master (root-commit) b53dd54] I added a file
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 hello.rb
```

Short message

- You can avoid opening your editor by stating the commit message with the -m option:

```
git commit –m 'This is my message'
```

SHA-1 of the commit

- Everything in Git uses SHA-1
  - (You'll get used to them, I promise!)
- 2^{160} possible hashes
- A unique identifier for our commit
- Use the first characters (at least four), if they're unique

The story so far

- The goal is to create a "commit"
- Each commit consists of one or more files, which can be in any directory
- Create a commit in two parts — first staging files ("git add") and then committing ("git commit")
- Each commit has a unique name, a 40-character SHA1

Hate the staging area?

- Many people do, especially if they're new to Git.
- Fortunately, there's a way around it!
- The "-a" flag means, "Stage all modified files, and commit them all."

```
git commit –a –m 'Commit everything'
```

Hate the staging area?

- You can also commit a single file:

```
git commit -m 'Commit everything' hello.rb
```

- However, this only works if a file is already known to Git, and has been modified!

How often to commit?

- Every time you have made a significant change.
- Fixed a bug? Commit.
- Fixed a typo? Commit.
- Changed a configuration file? Commit.
- Expect to have many (10-20) commits each day!
- The more commits, the fewer conflicts you'll have

Adding and committing

- That's what you'll be doing most of the time in Git
- You modify a file, add it to the staging area, and commit it
- Really, that's most of it!

Commit early and often

- Commits are the fundamental unit in Git.
- You *can* work on partial commits, but you really want to avoid doing so.
- Committing is cheap, easy to do, and should be done very often — say, 10-20 times each day.
  - Yes, really: 10-20 times each day.
- This allows you to revert to an earlier commit!

Many small commits

- Really, this is important!
- Don't wait a few days between commits. Rather, wait a few minutes or hours between commits.
- Most of the times I've seen people get into Git problems, it's because their commits are big and rare.
- Commit whenever you have finished a task of any sort, of any size, of any importance!

Adding parts of a file

- If you modified a file in several places, you can use "git add -p"