EPAM- Centre of Excellence

"Version Control with Git & GitHub"

Git Usage

C. Ravi Kishore Reddy

Assistant Professor

Department of CSE

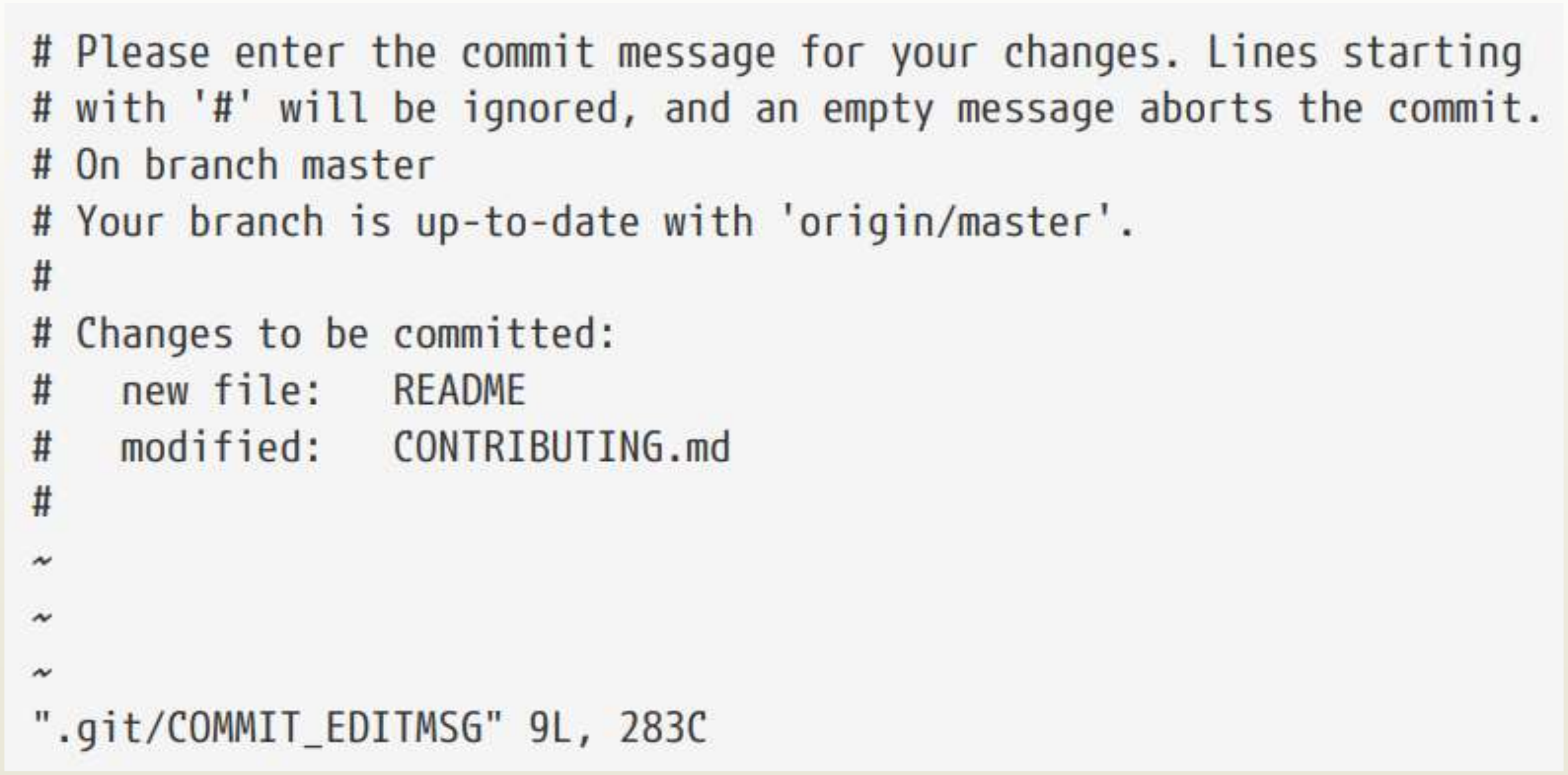
Contents

Git Basics

- **Committing Your Changes**
- Now that your staging area is set up the way you want it, you can commit your changes.
- Remember any files you have created or modified that you haven’t run git add on since you edited them — won’t go into this commit
- $ git commit

Git Basics

Committing Your Changes



```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is up-to-date with 'origin/master'.
#
# Changes to be committed:
#    new file:   README
#    modified:   CONTRIBUTING.md
#
~
~
~
".git/COMMIT_EDITMSG" 9L, 283C
```

Git Basics

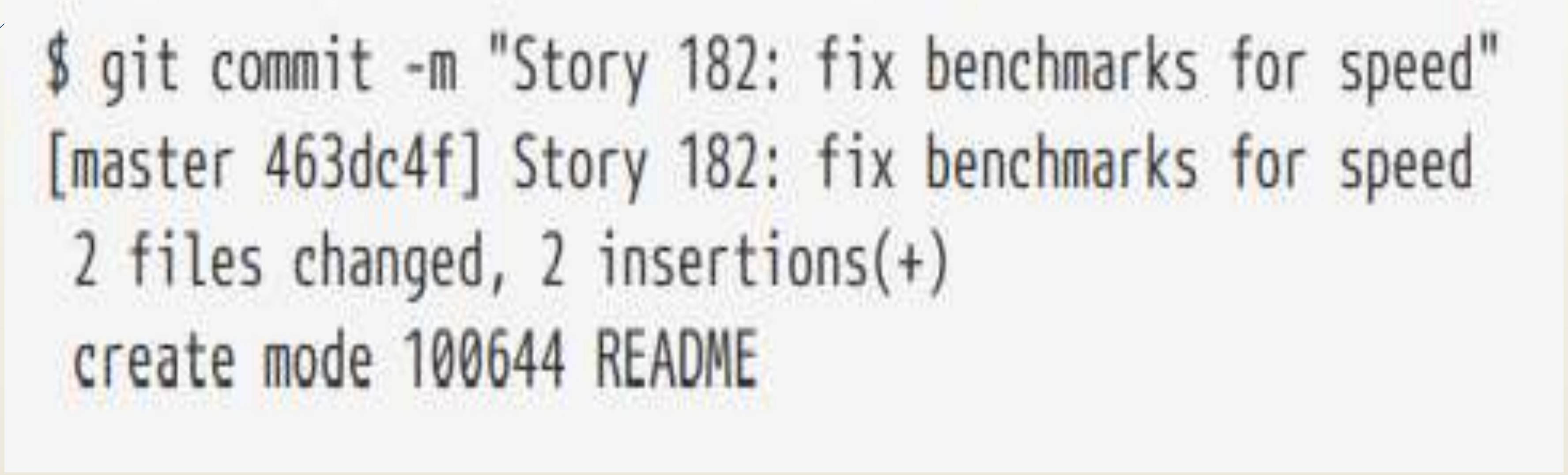
- **Committing Your Changes**
- You can see that the default commit message contains the latest output of the git status command commented out and one empty line on top.
- You can remove these comments and type your commit message, or you can leave them there to help you remember what you’re committing.
- When you exit the editor, Git creates your commit with that commit message.

Git Basics

- **Committing Your Changes**
- For an even more explicit reminder of what you’ve modified, you can pass the –v option to git commit.
- Doing so also puts the diff of your change in the editor so you can see exactly what changes you’re committing.

Git Basics

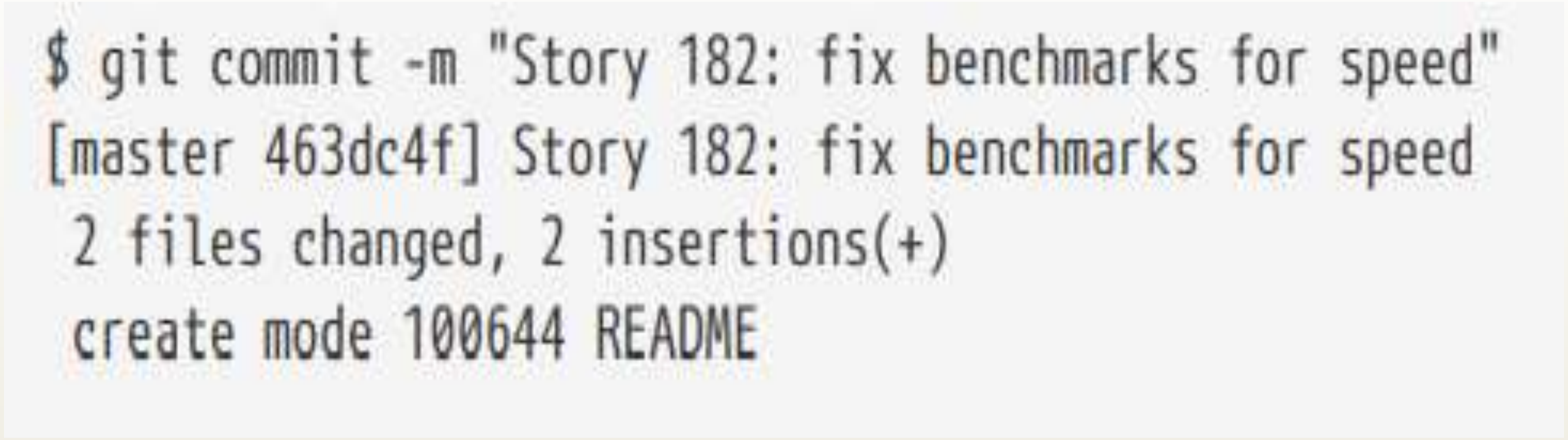
- **Committing Your Changes**
- Alternatively, you can type your commit message inline with the commit command by specifying it after a -m flag



```
$ git commit -m "Story 182: fix benchmarks for speed"
[master 463dc4f] Story 182: fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

Git Basics

- **Committing Your Changes**



```
$ git commit -m "Story 182: fix benchmarks for speed"
[master 463dc4f] Story 182: fix benchmarks for speed
 2 files changed, 2 insertions(+)
 create mode 100644 README
```

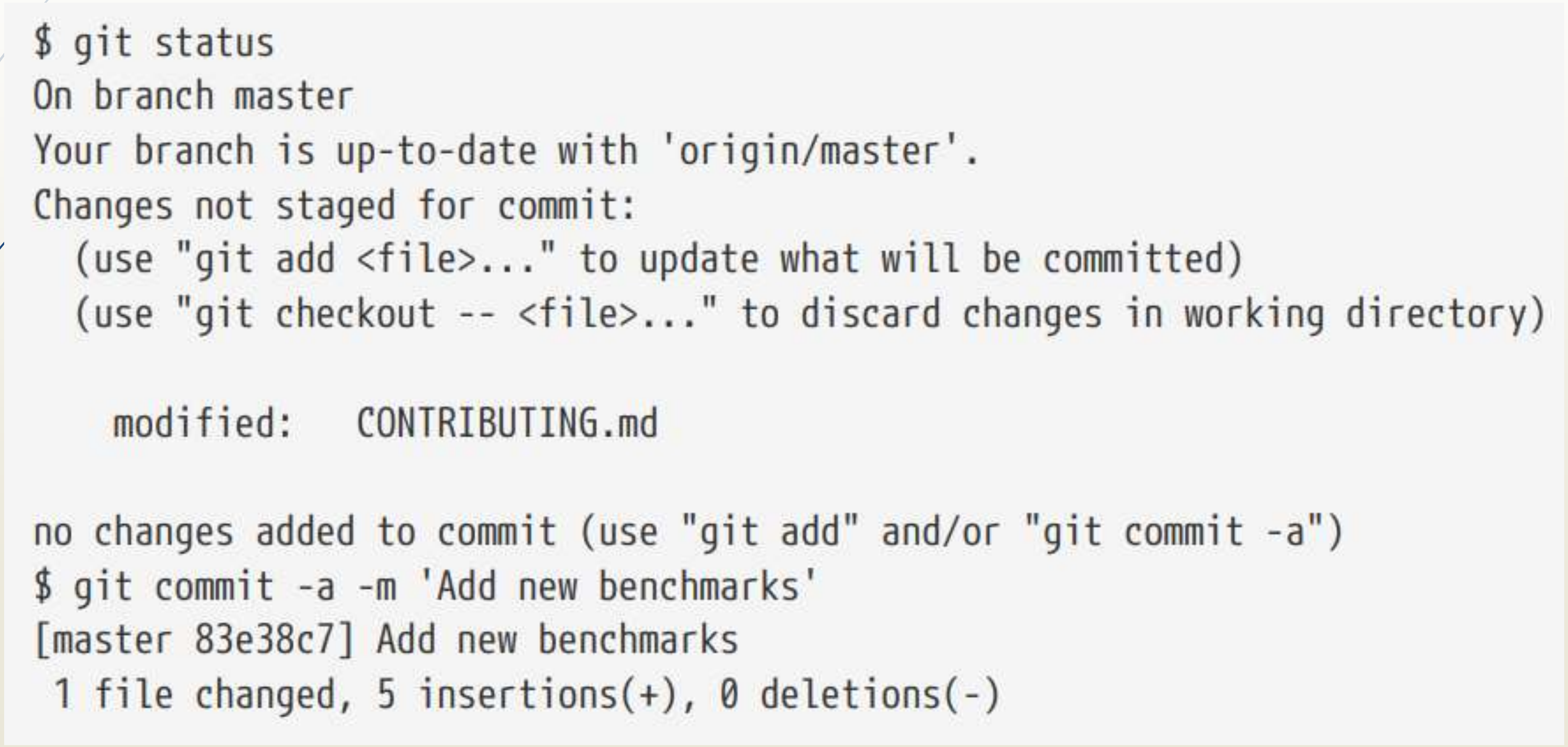
- You can see that the commit has given you some output about itself: which branch you committed to (master), what SHA-1 checksum the commit has (463dc4f), how many files were changed, and statistics about lines added and removed in the commit.

Git Basics

- **Skipping the staging area**
- If you want to skip the staging area, Git provides a simple shortcut.
- Adding the -a option to the *git commit* command makes Git automatically stage every file that is already tracked before doing the commit, letting you skip the *git add* part

Git Basics

- **Skipping the staging area**



```
$ git status
On branch master
Your branch is up-to-date with 'origin/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:   CONTRIBUTING.md

no changes added to commit (use "git add" and/or "git commit -a")
$ git commit -a -m 'Add new benchmarks'
[master 83e38c7] Add new benchmarks
 1 file changed, 5 insertions(+), 0 deletions(-)
```

Git Basics

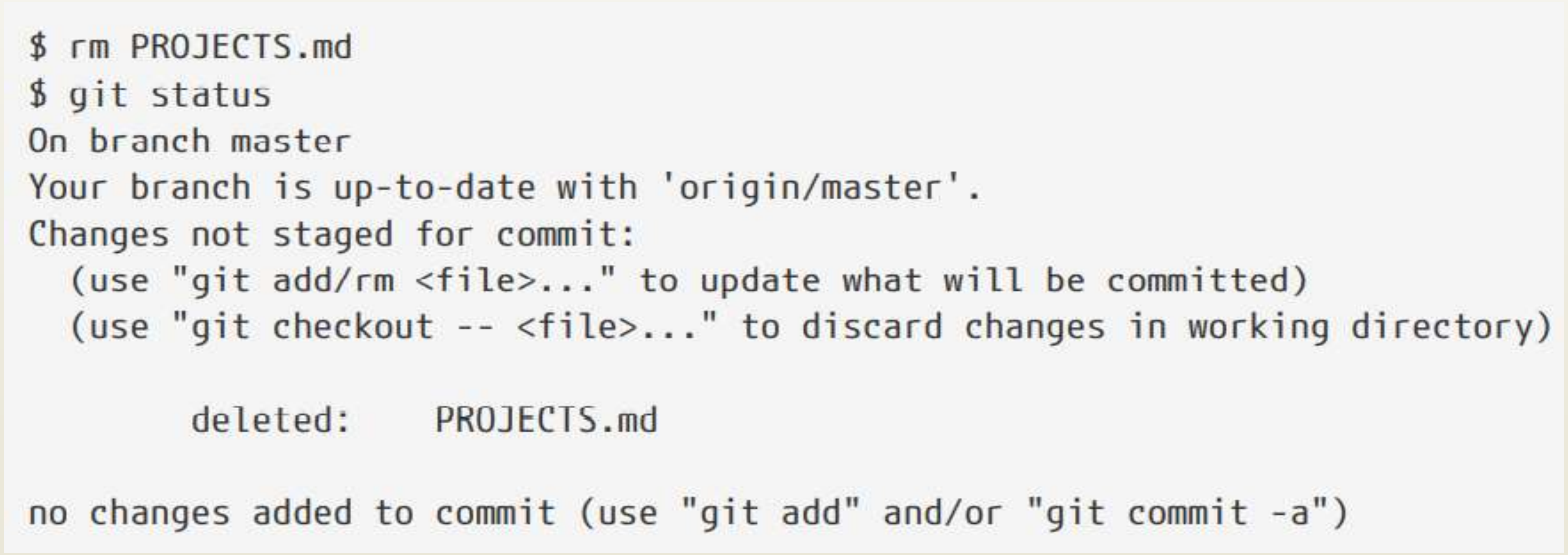
- **Untracking the files**
- You may want to do is to keep the file in your working tree but remove it from your staging area.
- You may want to keep the file on your hard drive but not have Git track it anymore.
- This is particularly useful if you forgot to add something to your .gitignore file.
- $ git rm --cached README

Git Basics

- **Removing Files**
- To remove a file from Git, you have to remove it from your tracked files (more accurately, remove it from your staging area) and then commit.
- The *git rm* command does that, and also removes the file from your working directory so you don’t see it as an untracked file the next time around.

Git Basics

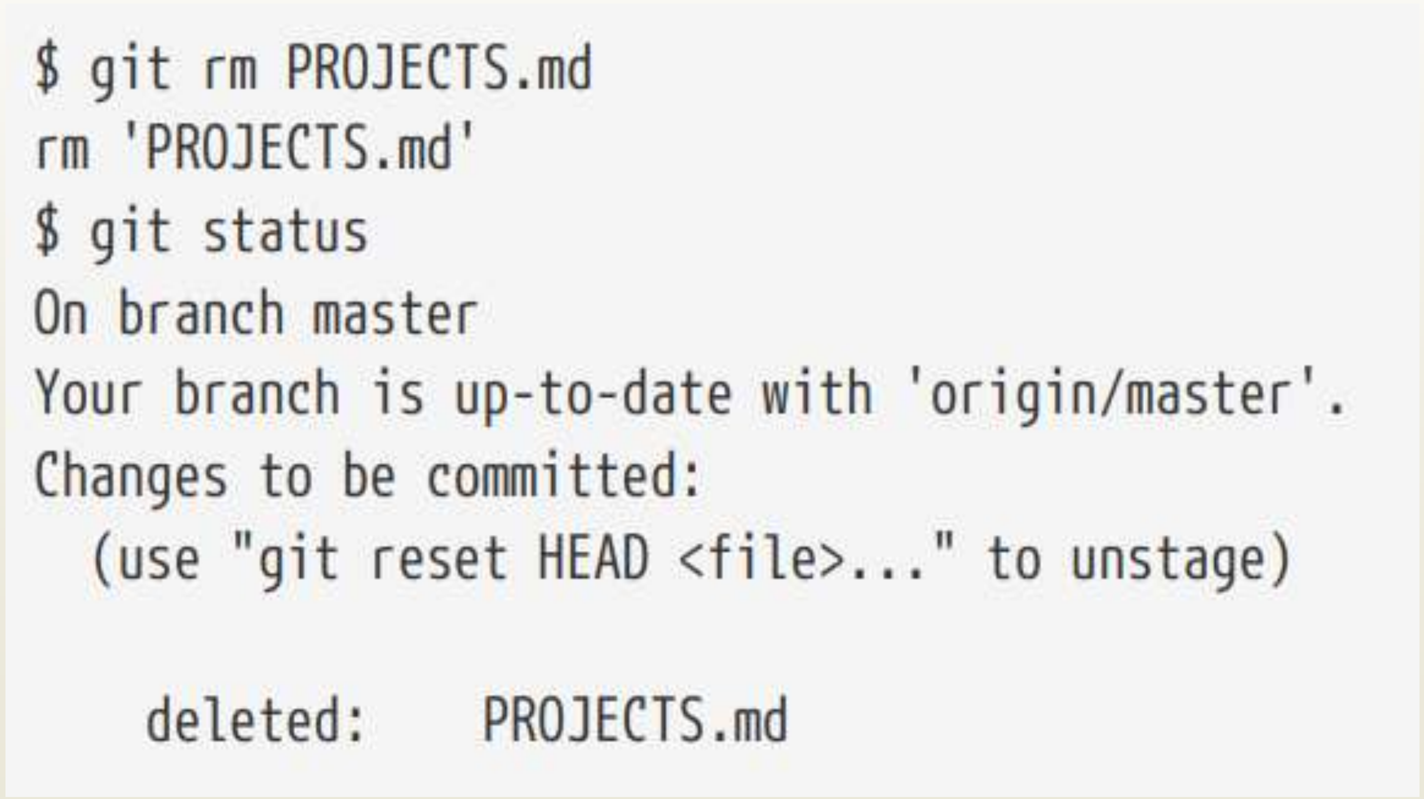
- **Removing Files**
- If you simply remove the file from your working directory, it shows up under the “Changes not staged for commit” (that is, unstaged) area of your git status output



```
$ rm PROJECTS.md
$ git status
On branch master
Your branch is up-to-date with 'origin/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:    PROJECTS.md

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

Git Basics

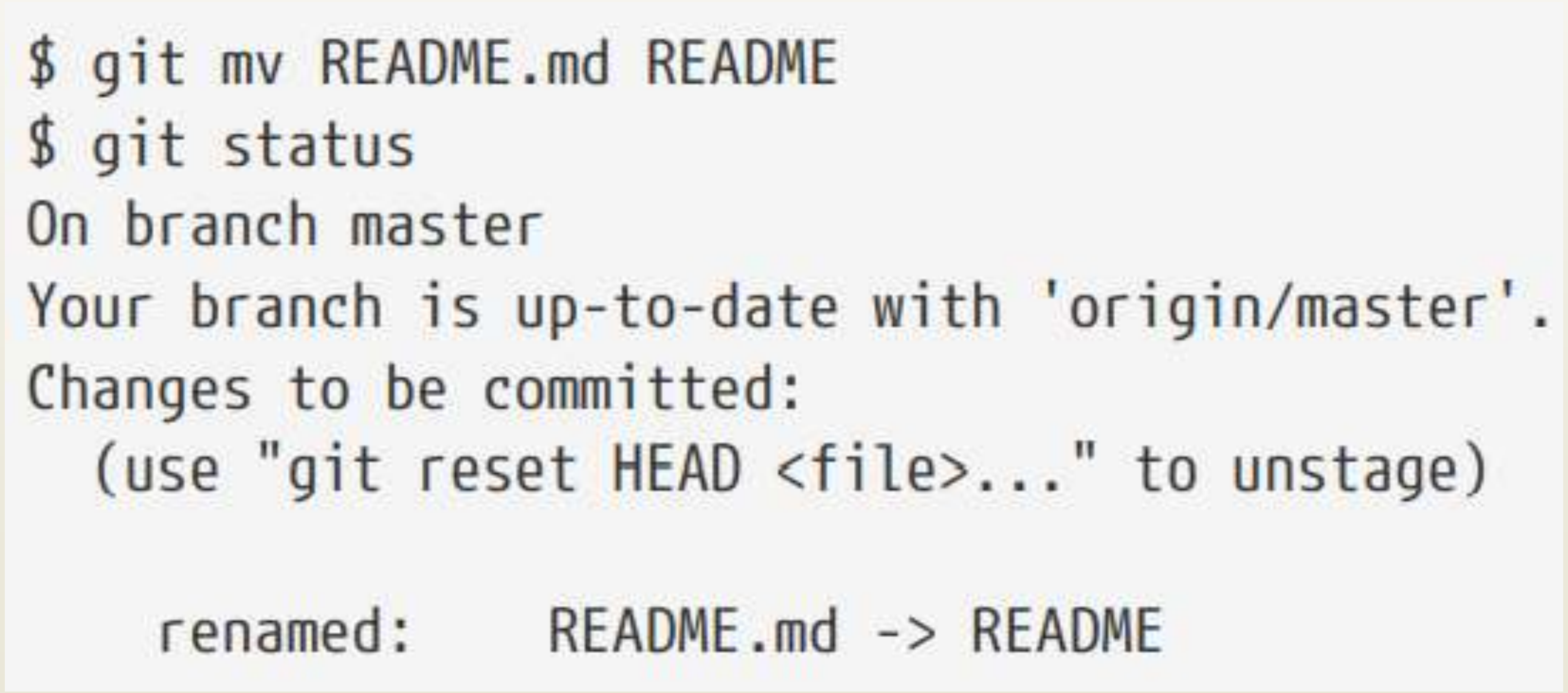
- **Removing Files**
- If you run *git rm*, it stages the file’s removal



```
$ git rm PROJECTS.md
rm 'PROJECTS.md'
$ 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)

    deleted:    PROJECTS.md
```

Git Basics

- **Moving Files**
- Git has a mv command. If you want to rename a file in Git
- $ git mv file_from file_to



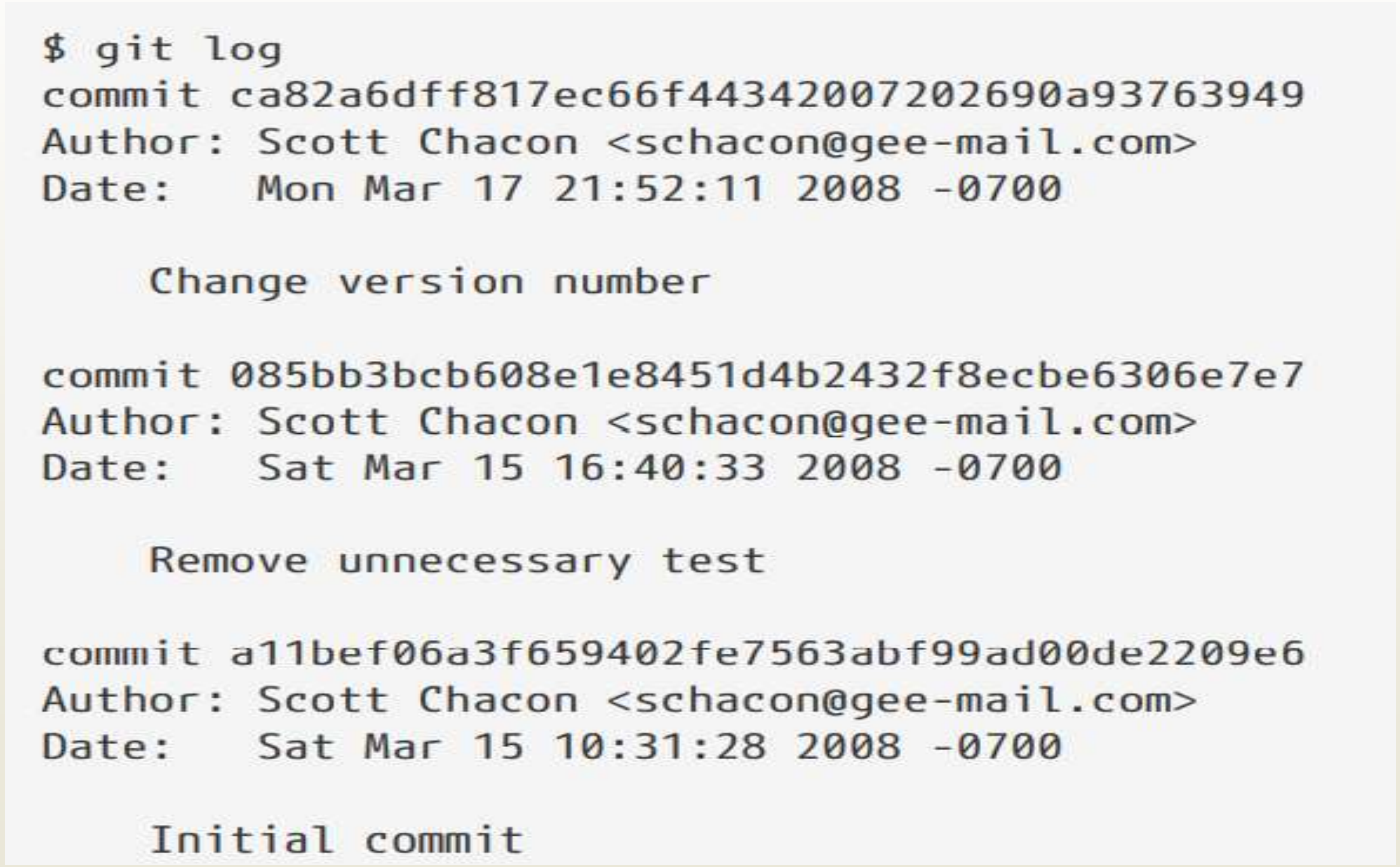
```
$ git mv README.md README
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

Git Basics

- **Viewing the Commit History**
- After you have created several commits you'll probably want to look back to see what has happened.
- The most basic and powerful tool to do this is the git log command.
- By default, with no arguments, git log lists the commits made in that repository in reverse chronological order;
- Each commit with its SHA-1 checksum, the author's name and email, the date written, and the commit message.

Git Basics

Viewing the Commit History



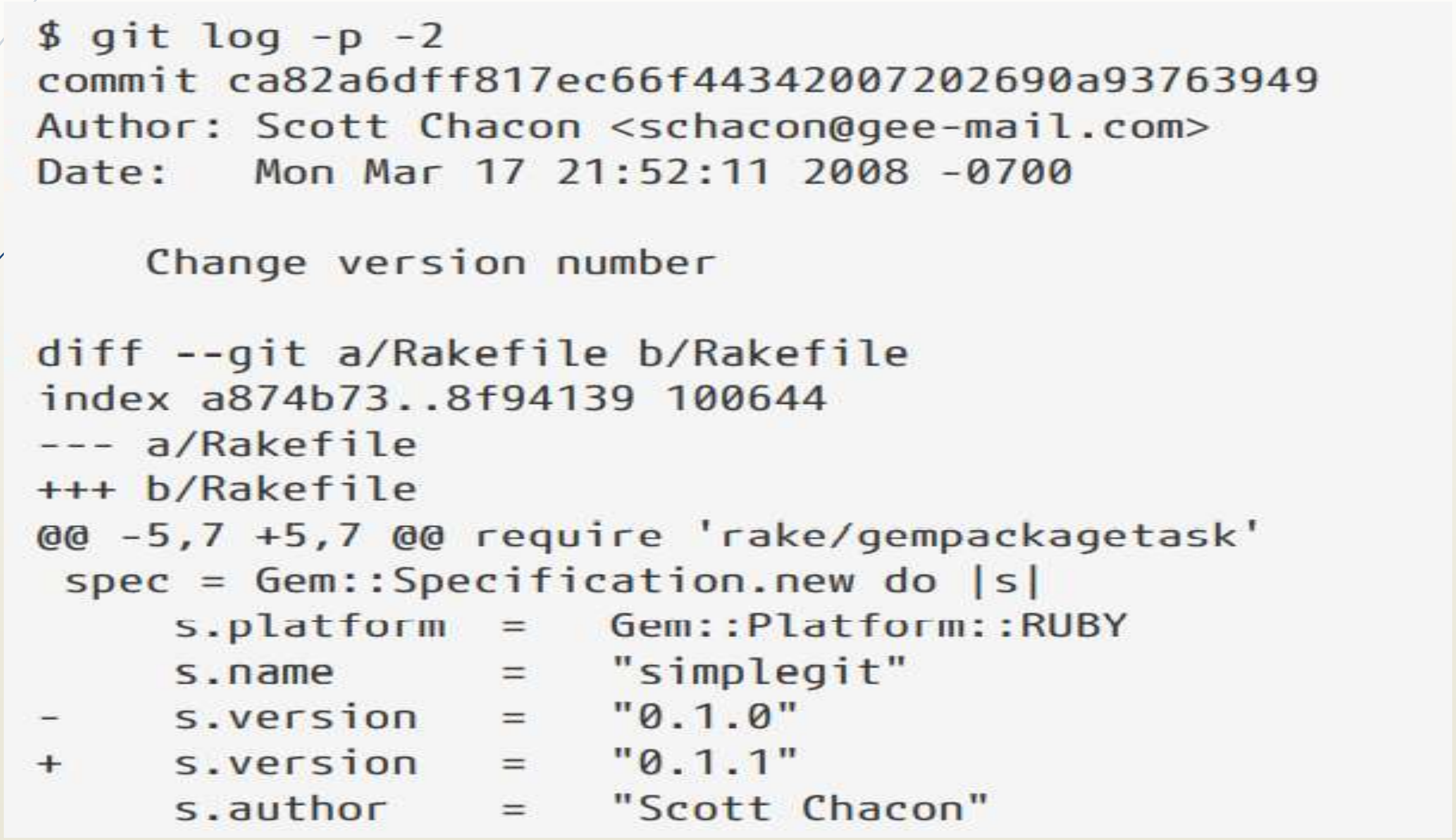
```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    Initial commit
```

Git Basics

- **Viewing the Commit History**
- Huge number of options exist for git log command
- One of the more helpful options is -p or --patch, which shows the difference (the patch output) introduced in each commit.
- You can also limit the number of log entries displayed, such as using -2 to show only the last two entries.

Git Basics

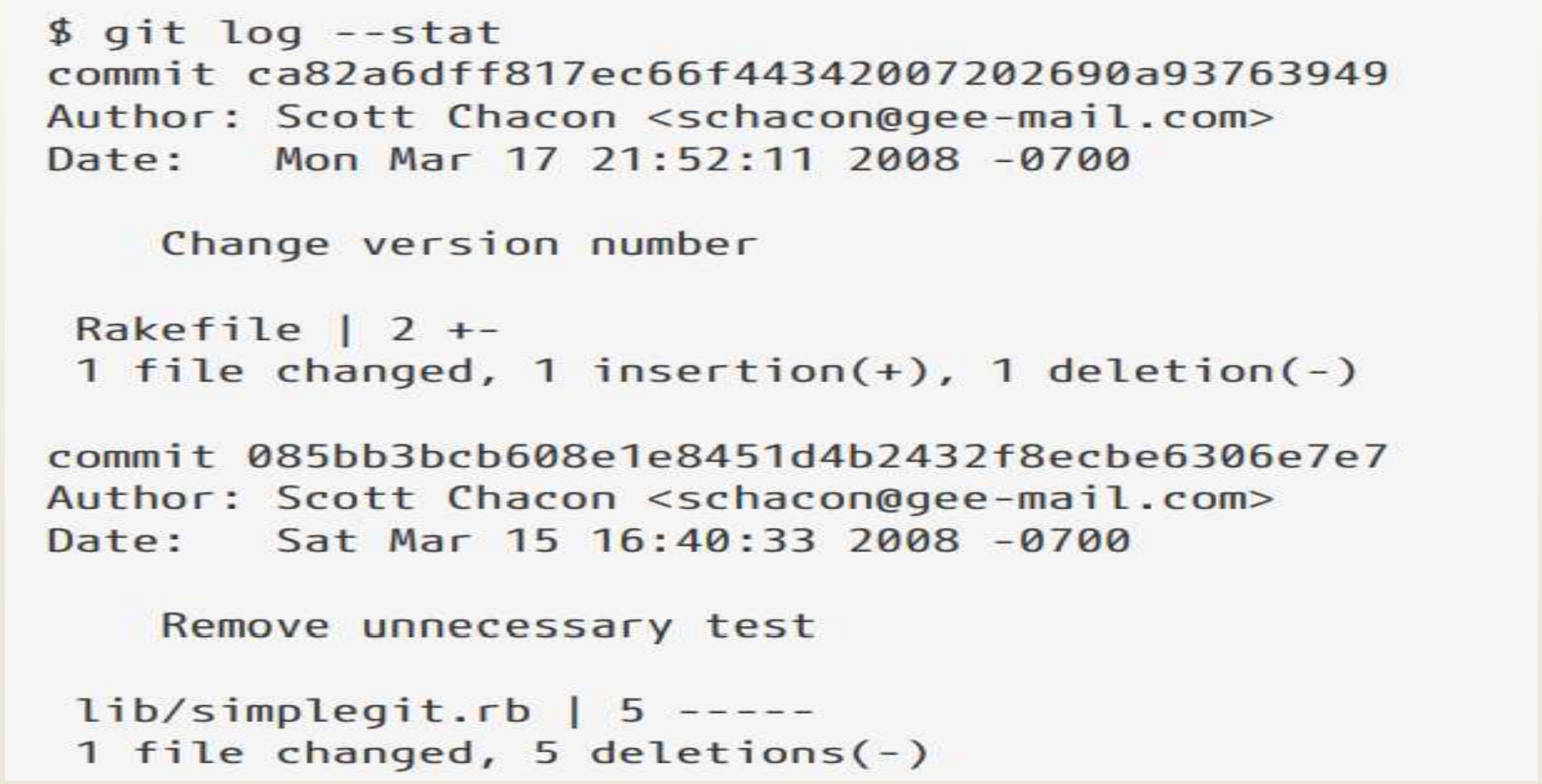
Viewing the Commit History



```
$ git log -p -2
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

diff --git a/Rakefile b/Rakefile
index a874b73..8f94139 100644
--- a/Rakefile
+++ b/Rakefile
@@ -5,7 +5,7 @@ require 'rake/gempackagetask'
 spec = Gem::Specification.new do |s|
     s.platform  =   Gem::Platform::RUBY
     s.name      =   "simplegit"
-    s.version   =   "0.1.0"
+    s.version   =   "0.1.1"
     s.author    =   "Scott Chacon"
```

Git Basics

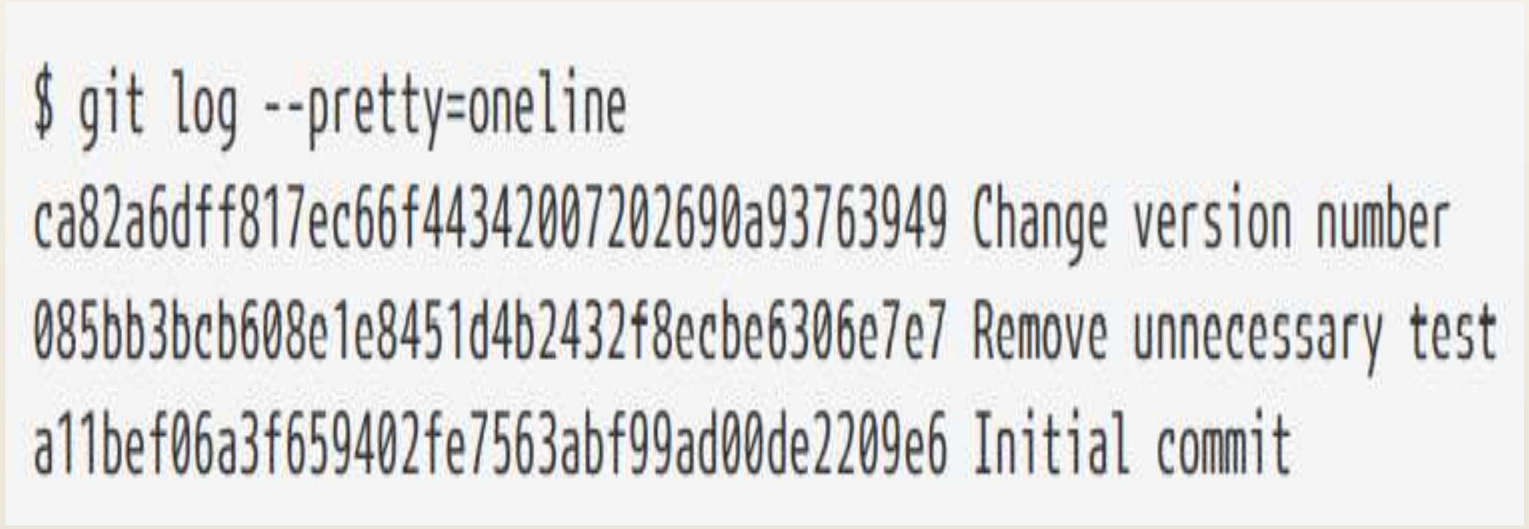
- **Viewing the Commit History**
- If you want to see some abbreviated stats for each commit, you can use the --stat option



```
$ git log --stat
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    Change version number

 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    Remove unnecessary test

 lib/simplegit.rb | 5 -----
 1 file changed, 5 deletions(-)
```

Git Basics

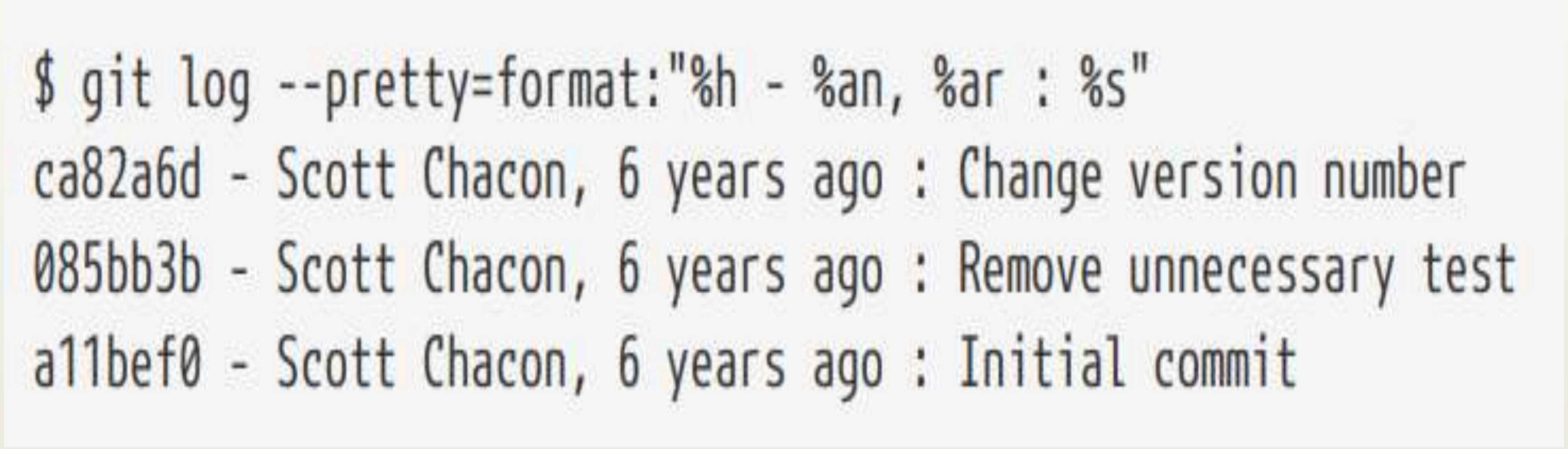
- **Viewing the Commit History**
- Another really useful option is --pretty. This option changes the log output to formats other than the default.
- The oneline value for this option prints each commit on a single line



```
$ git log --pretty=oneline
ca82a6dff817ec66f44342007202690a93763949 Change version number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 Remove unnecessary test
a11bef06a3f659402fe7563abf99ad00de2209e6 Initial commit
```

Git Basics

- **Viewing the Commit History**
- The most interesting option value is format, which allows you to specify your own log output format.



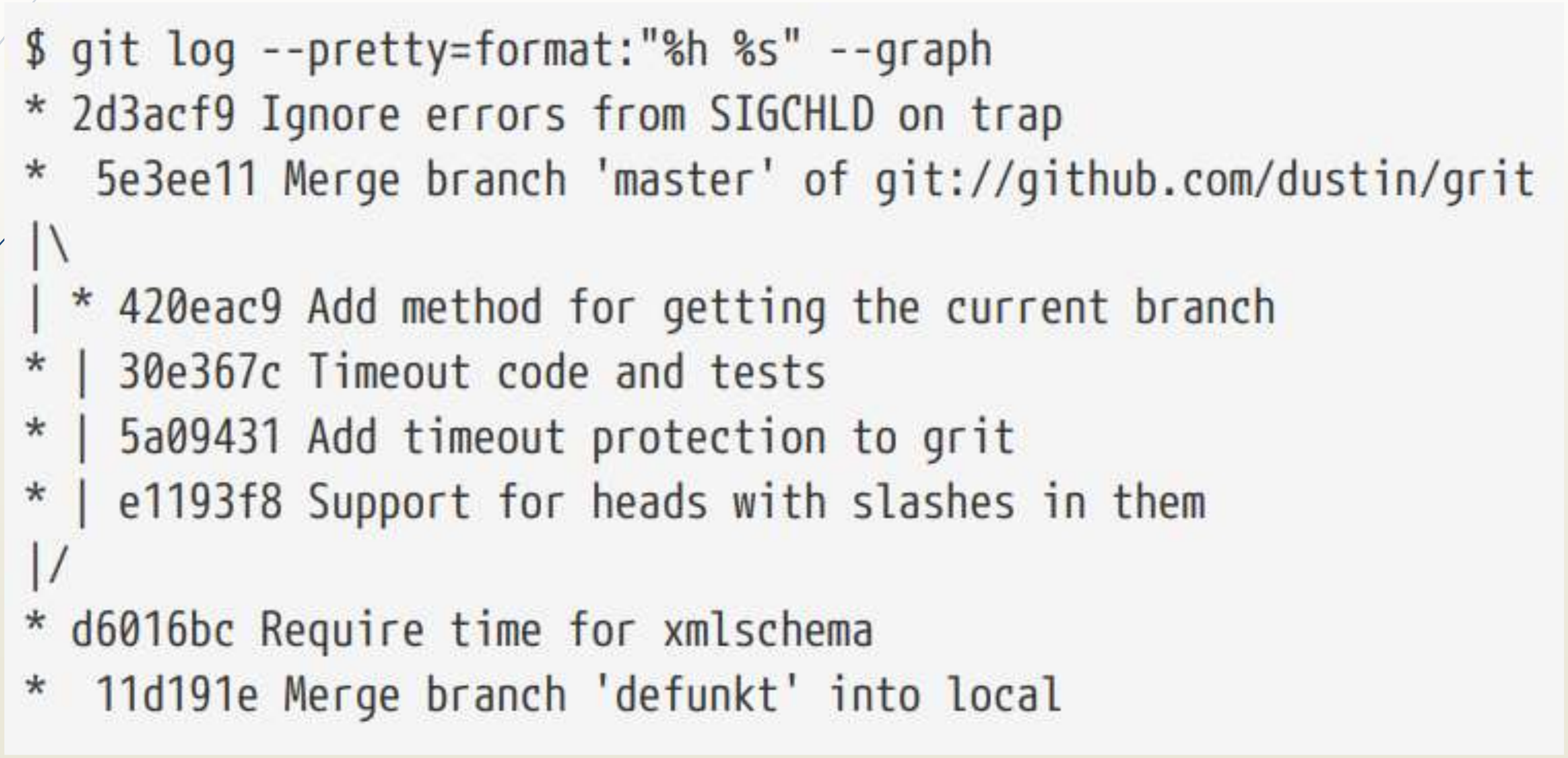
```
$ git log --pretty=format:"%h - %an, %ar : %s"
ca82a6d - Scott Chacon, 6 years ago : Change version number
085bb3b - Scott Chacon, 6 years ago : Remove unnecessary test
a11bef0 - Scott Chacon, 6 years ago : Initial commit
```

Git Basics

- **Viewing the Commit History**
- The oneline and format option values are particularly useful with another log option called --graph.
- This option adds a nice little ASCII graph showing your branch and merge history:

Git Basics

Viewing the Commit History



```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 Ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Add method for getting the current branch
* | 30e367c Timeout code and tests
* | 5a09431 Add timeout protection to grit
* | e1193f8 Support for heads with slashes in them
|/
* d6016bc Require time for xmlschema
*  11d191e Merge branch 'defunkt' into local
```

Git Basics

- **Undoing Things**
- At any stage, you may want to undo something.
- Be careful, because you can’t always undo some of these undos.
- This is one of the few areas in Git where you may lose some work if you do it wrong.

Git Basics

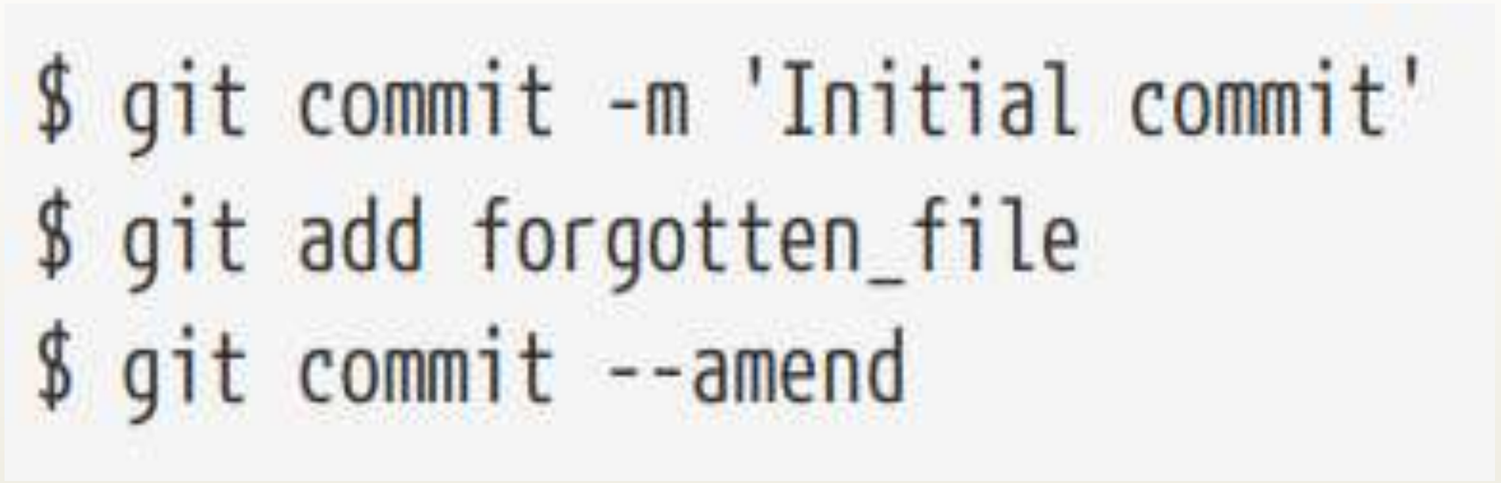
- **Undoing Things**
- One of the common undos takes place when you commit too early and possibly forget to add some files, or you mess up your commit message.
- If you want to redo that commit, make the additional changes you forgot, stage them, and commit again using the --amend option
- $ git commit --amend

Git Basics

- **Undoing Things**
- `$ git commit --amend`
- This command takes your staging area and uses it for the commit.
- The same commit-message editor fires up, but it already contains the message of your previous commit.
- You can edit the message the same as always, but it overwrites your previous commit.

Git Basics

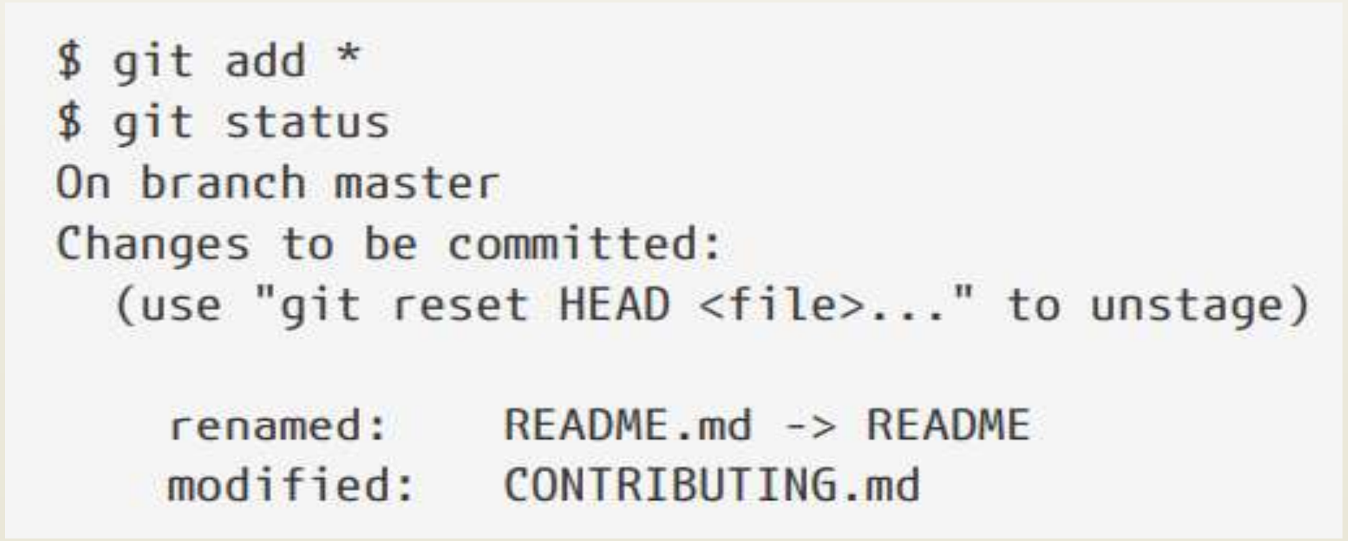
- **Undoing Things**



```
$ git commit -m 'Initial commit'
$ git add forgotten_file
$ git commit --amend
```

Git Basics

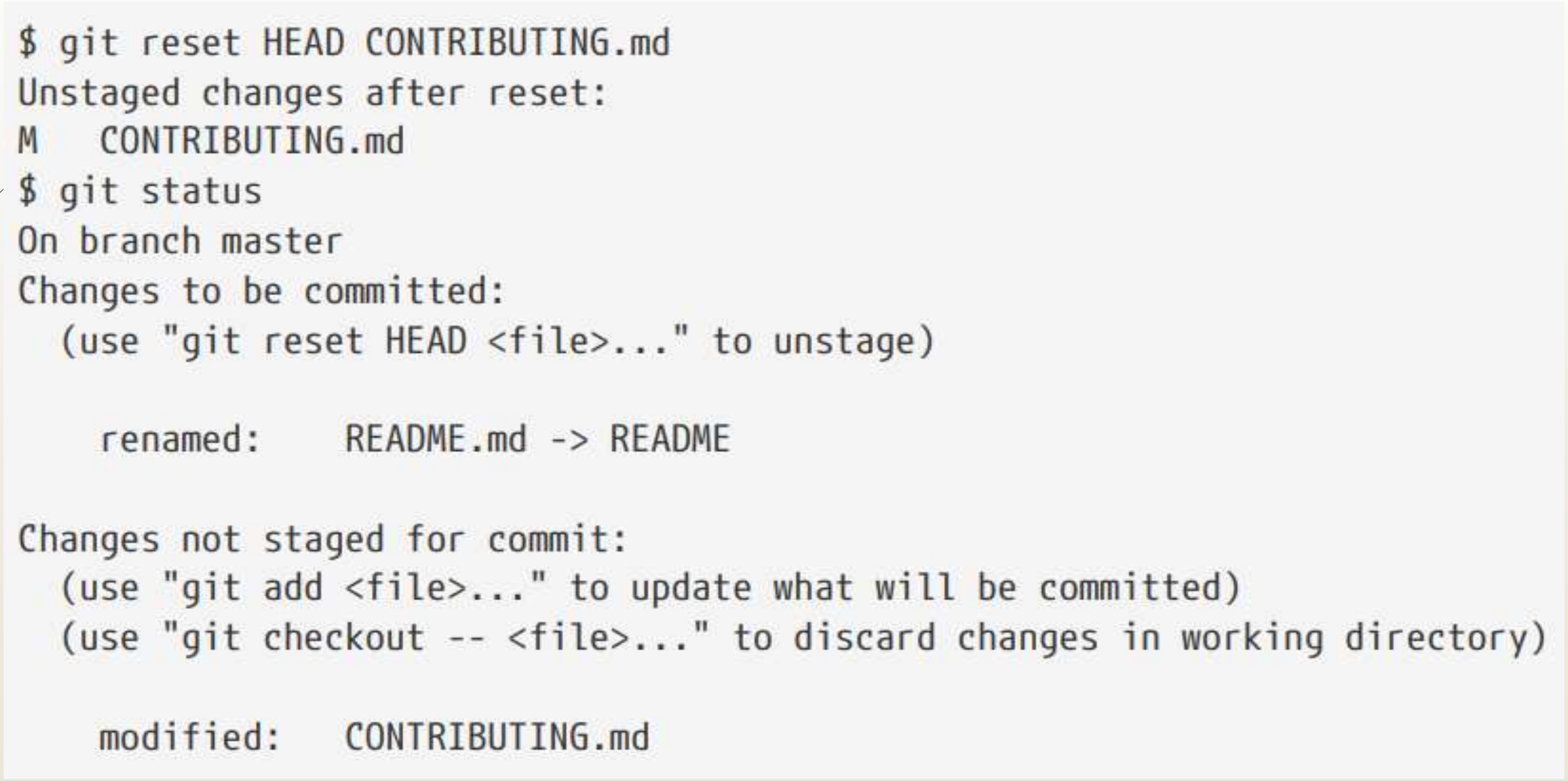
- **Unstaging a Staged File**
- For example, let’s say you’ve changed two files and want to commit them as two separate changes, but you accidentally type git add * and stage them both. How can you unstage one of the two?



```
$ git add *
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
    modified:   CONTRIBUTING.md
```

Git Basics

Unstaging a Staged File



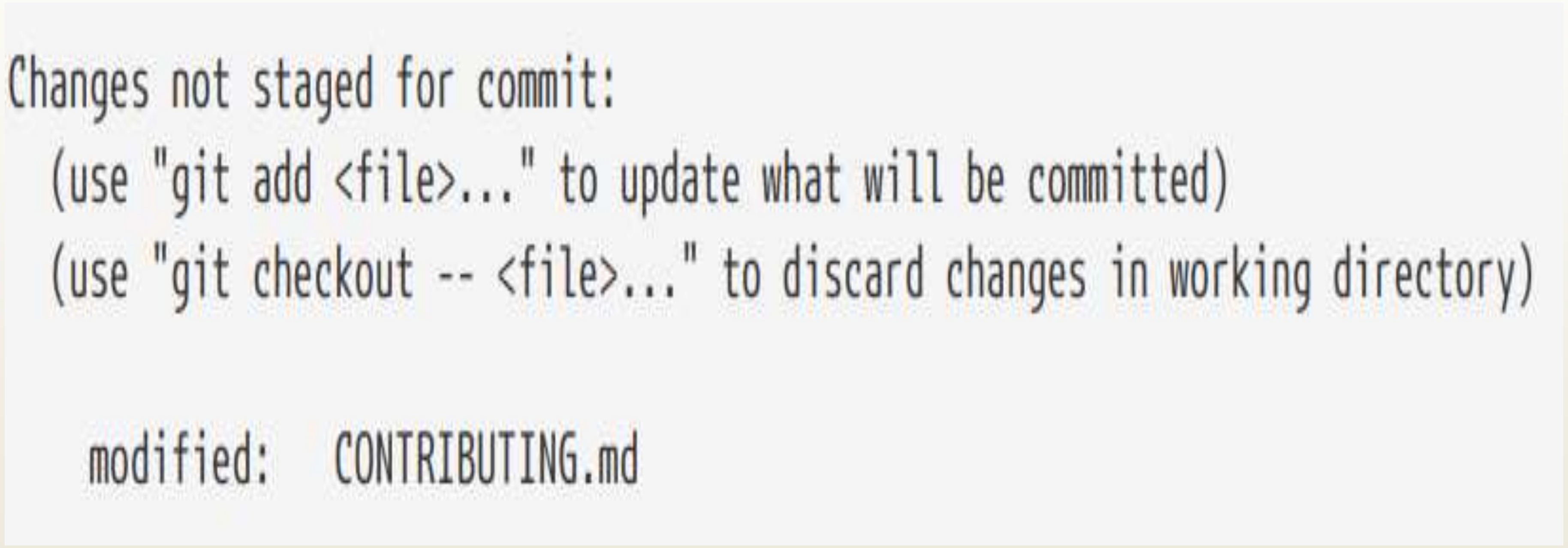
```
$ git reset HEAD CONTRIBUTING.md
Unstaged changes after reset:
M    CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README

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:   CONTRIBUTING.md
```

Git Basics

- **Unmodifying a Modified File**
- What if you realize that you don’t want to keep your changes to the CONTRIBUTING.md file?
- How can you easily unmodify it — revert it back to what it looked like when you last committed?
- *git status* tells you how to do that

Git Basics

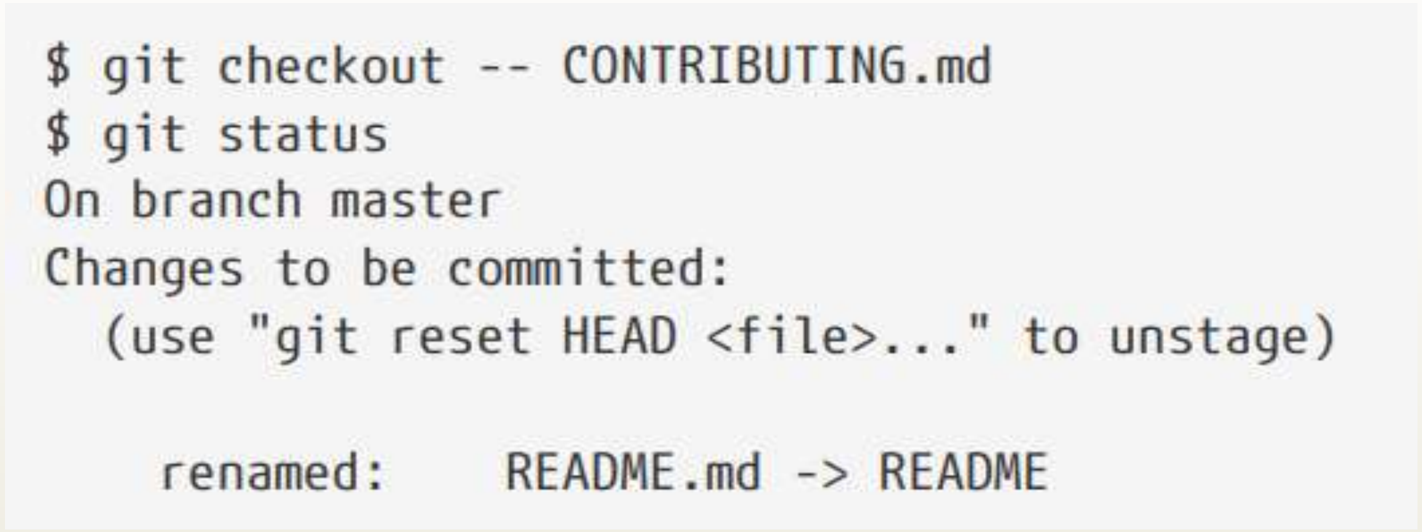
- **Unmodifying a Modified File**



```
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:   CONTRIBUTING.md
```

Git Basics

Unmodifying a Modified File



```
$ git checkout -- CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    README.md -> README
```

- It’s important to understand that git checkout -- <file> is a dangerous command. Any local changes you made to that file are gone — Git just replaced that file with the last staged or committed version. Don’t ever use this command unless you absolutely know that you don’t want those unsaved local changes

Git Basics

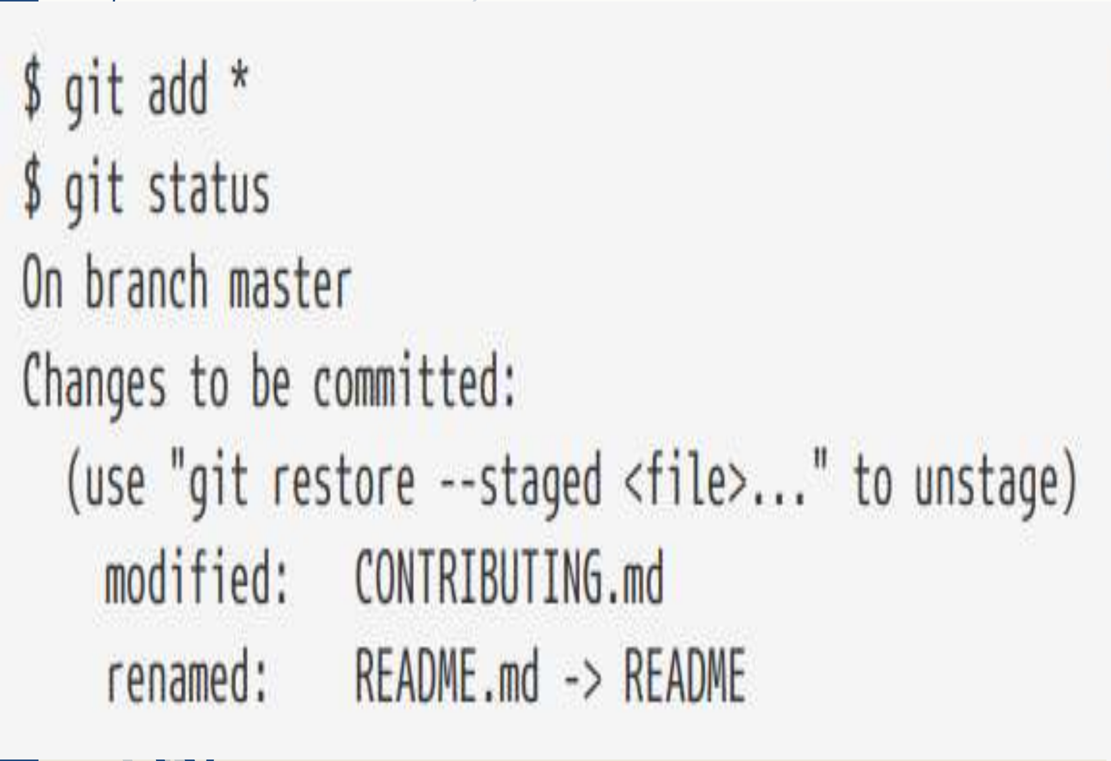
- **Unmodifying a Modified File**
- Remember, anything that is committed in Git can almost always be recovered.
- Even commits that were on branches that were deleted or commits that were overwritten with an --amend commit can be recovered.
- However, anything you lose that was never committed is likely never to be seen again

Git Basics

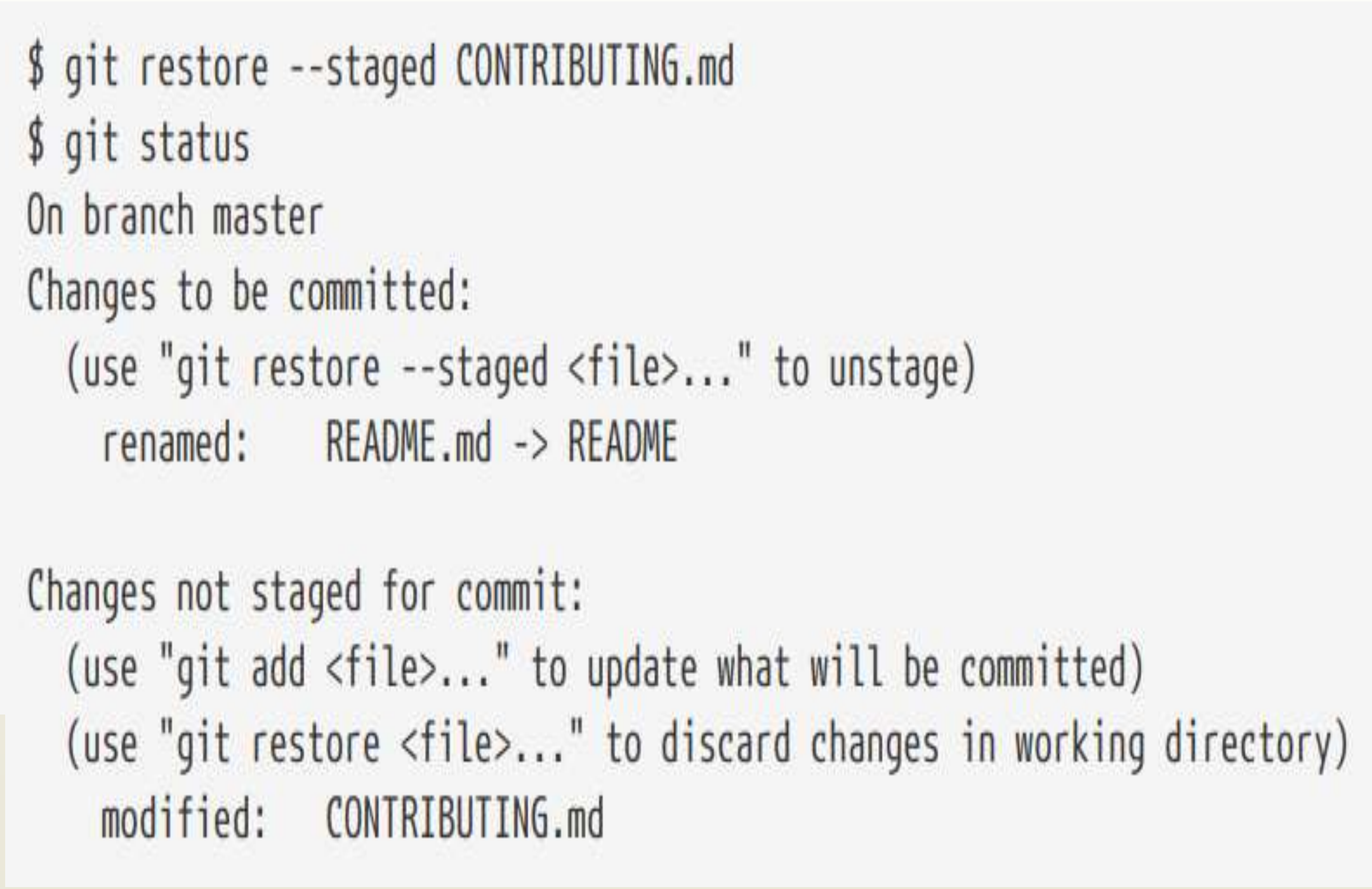
- **Undoing things with git restore**
- Git version 2.23.0 introduced a new command: *git restore*.
- It's basically an alternative to *git reset*.
- From Git version 2.23.0 onwards, Git will use git restore instead of git reset for many undo operations.

Git Basics

- **Unstaging a Staged File with git restore**



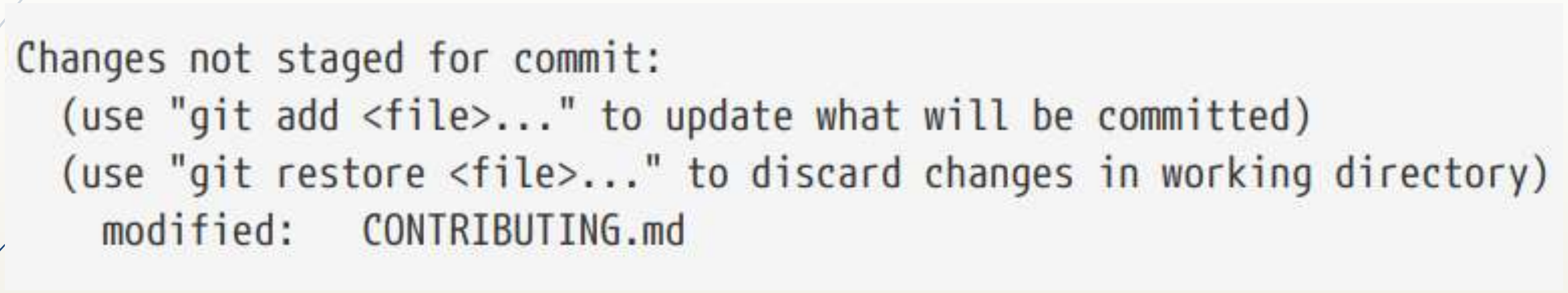
```
$ git add *
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   CONTRIBUTING.md
    renamed:    README.md -> README
```



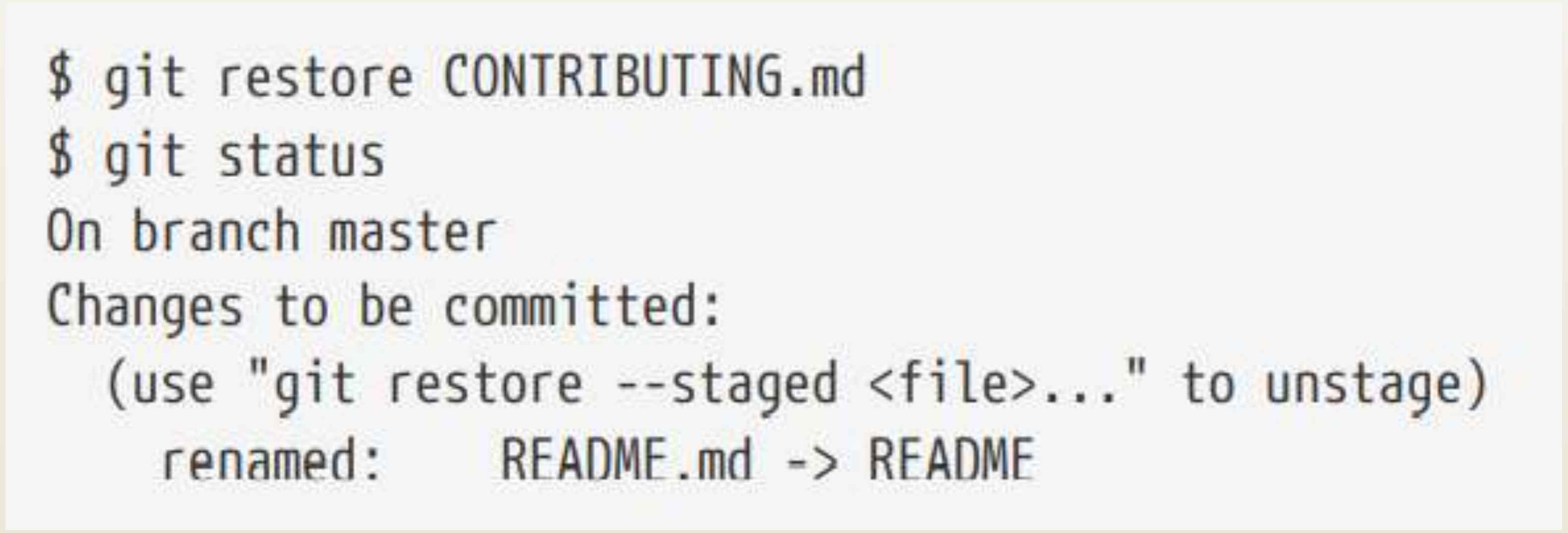
```
$ git restore --staged CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    README.md -> README

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   CONTRIBUTING.md
```

Git Basics

Unmodifying a Modified File with git restore



```
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   CONTRIBUTING.md
```



```
$ git restore CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    renamed:    README.md -> README
```

Thank You