Getting Started with Git

Dennis Jarecke
Six Sycamores, LLC
November 15, 2014

Who am I?

Presentation available at

www.linkedin.com/in/dennisjarecke

What is Git?

Git is a distributed revision control and source code management system with an emphasis on speed, data integrity, and support for distributed, non-linear workflows.

It was originally created by Linus Torvalds for Linux development, but has since become the most widely adopted version control system for code development.

Source: en.wikipedia.org/wiki/Git_(software)

A Highly Simplistic Overview

A git repository is a directory or folder on your computer with the files you want to track and a .git directory which holds configurations settings, branches, commits, etc.

Each repository is made up of one or many branches.

Branches are made up of commits. Commits are a series of changes made to the files in the folder.

Branches can track branches on other computers - this is why a central server is not required.

More Detail on the Repository

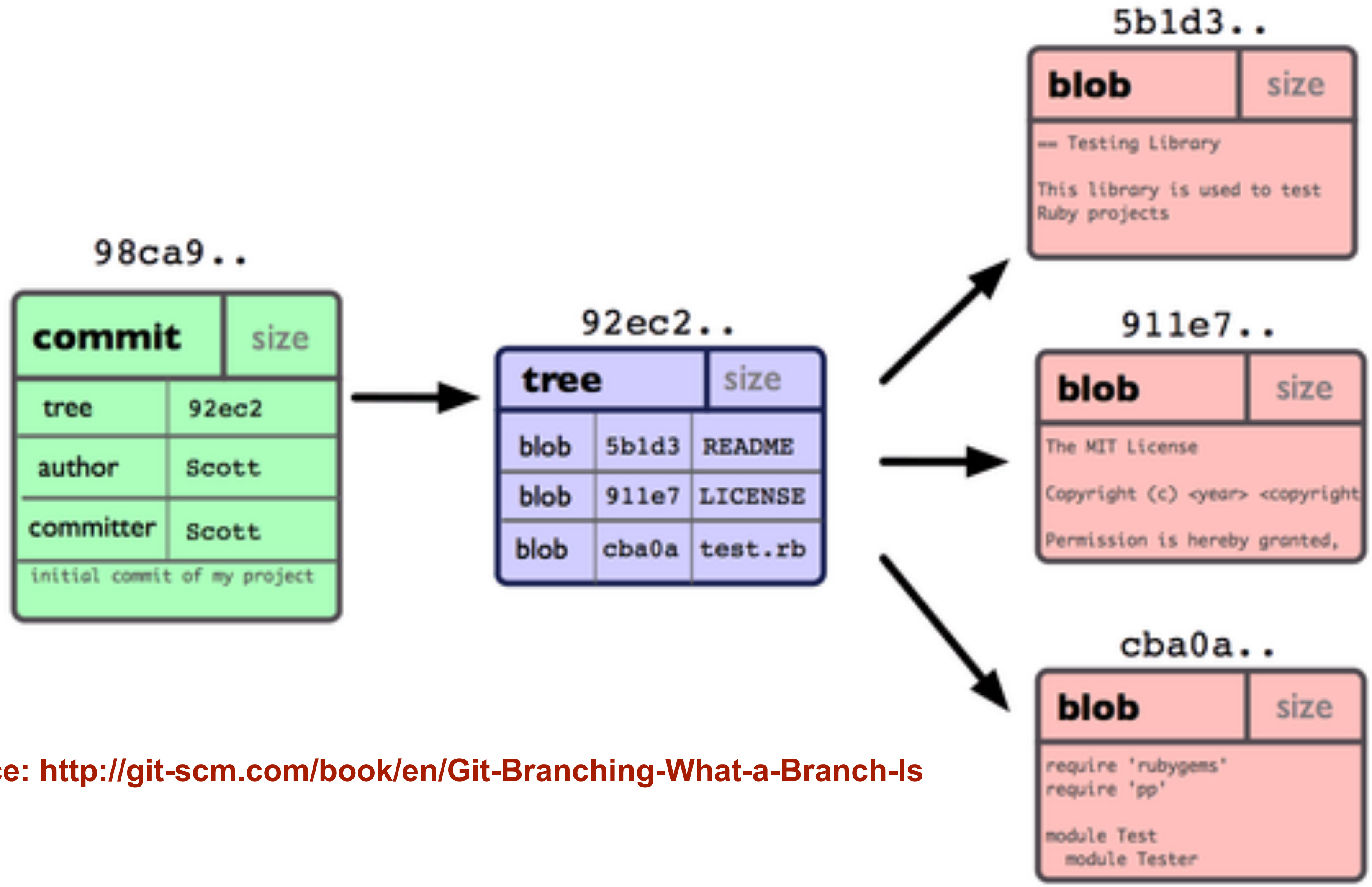
- Git repository - database of changes, revisions, history, etc. It has three components
- Configuration settings - local to that repository only
- Two primary data structures
  - Index - private to a repository
  - Object Store - permanent changes that get copied or cloned

4 Objects in Object Store

- ❖ Blob: Each version of a file is represented as a blob. There is no metadata in the blob including the filename!
- ❖ Tree: one level of directory info including blob identifiers, path names, metadata, other trees. This recreates the directory structure.
- ❖ Commit: metadata about a change - author, log message date, etc. It points to a tree object and has a commit parent - thus a full history can be constructed.
- ❖ Tag: assigns a human-readable name to an object - usually a commit.

SHA1 = Object IDs

- Every object in the object store has a unique SHA1 hash value.
- See http://en.wikipedia.org/wiki/SHA-1
- ANY change to an object will cause a change in its SHA1 hash value - thus ANY change to a file will tell git (and you) that the file has changed.
- SHA1s are globally unique - all repositories will have the same SHA1 for the same file, path, etc. i.e. the same SHA1 for identical blobs, trees, commits, tags.
- Git speak: SHA1, hash code, object id used interchangeably.



Source: http://git-scm.com/book/en/Git-Branching-What-a-Branch-Is

The Index

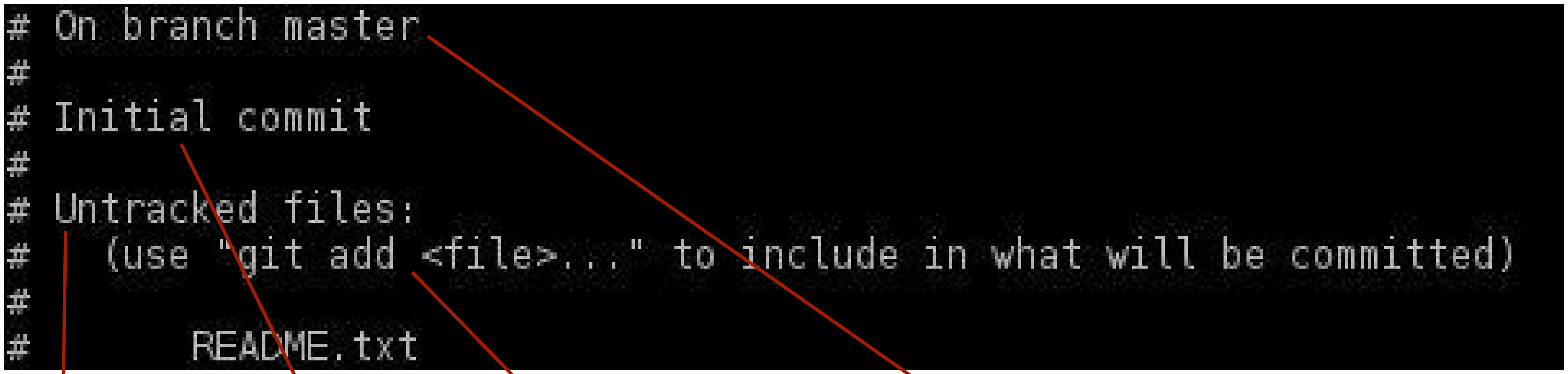
- A temporary and dynamic binary file that contains a snapshot of the folder your repository is in at some point in time.
- An index can be a specific commit or can be changes you will commit later.
- The index records changes you have made to a file.
- Specific changes in an index will later become a permanent commit in the repository.

Git Exercise 1:

- Move to the folder where you want to create a new git repository - it doesn’t matter if there are files there or not. Let’s do an empty one as a beginning example.
- *git init* - will create a .git folder and initialize the git repository. This is only done once.
- Create README.txt with the line “Hello world!”

Git Exercise 1 (cont'd):

- *git status* will now show an untracked file.



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

Untracked files

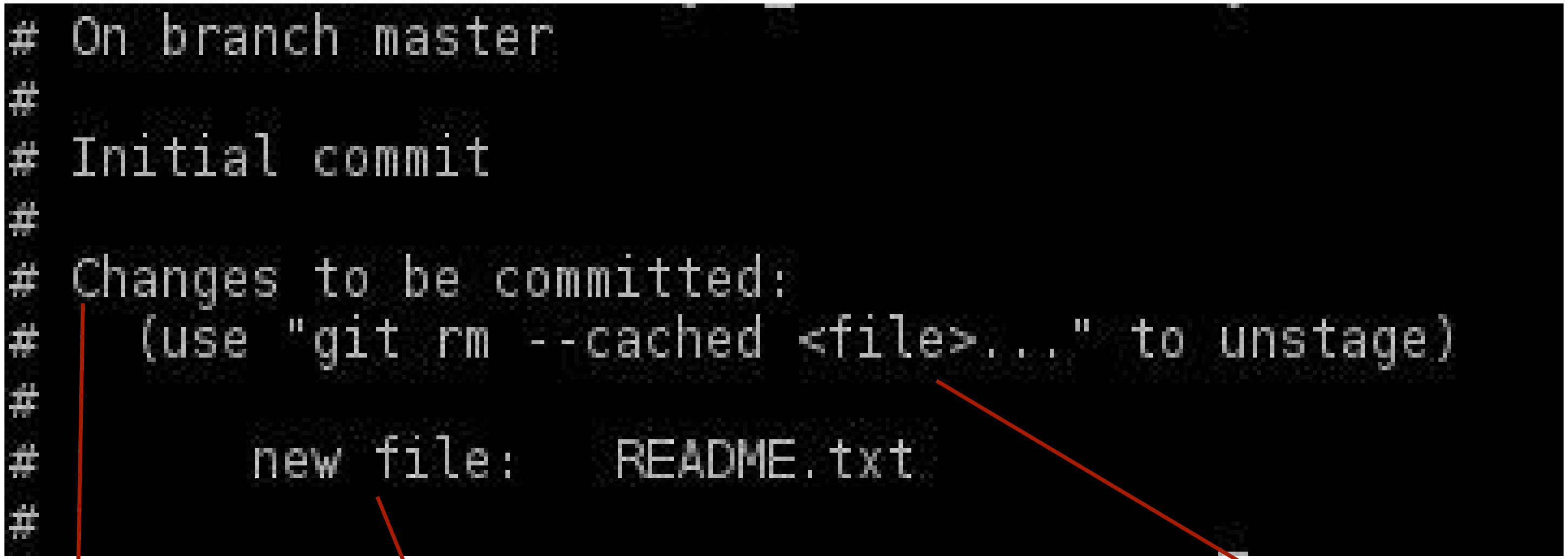
You haven't put anything into the repository yet.

A prompt of what to do.

VERY IMPORTANT!
master is the "default" branch in git. However, it is not unique or special. You can delete it and use another branch that you create.

Git Exercise 1 (cont'd):

- *git add README.txt* will add this to the index. Then type *git status* again.



```
# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   README.txt
#
```

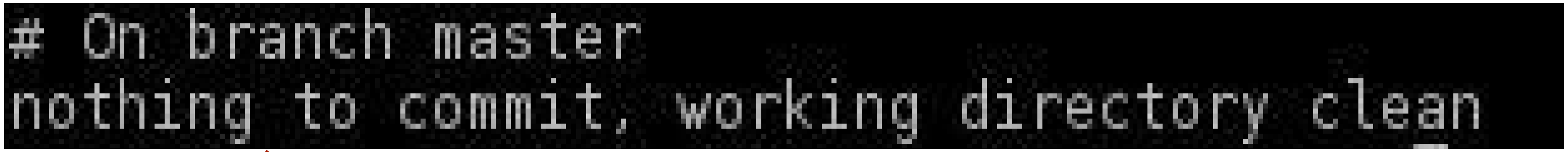
It tells you the files that need to be committed.

Yes, the file is new and has never been committed.

It tells you the exact command if you want to remove it from the index and not have the file and its changes committed.

Git Exercise 1 (cont'd):

- *git commit -m "initial commit"* will commit this change so it is permanently stored. Then type *git status* again.

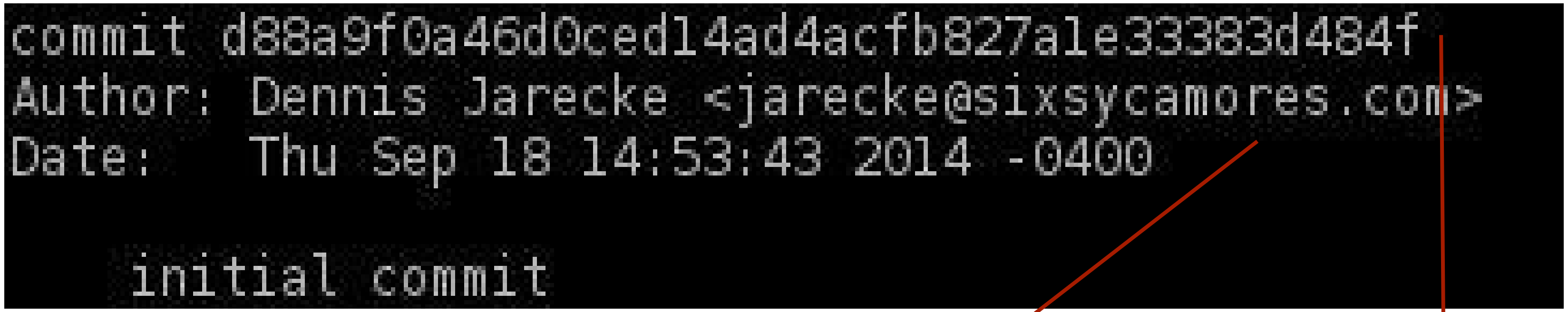


```
# On branch master
nothing to commit, working directory clean
```

Congratulations! The README.txt file has been committed!

Git Exercise 1 (cont'd):

- *git log* will show the commit logs



```
commit d88a9f0a46d0ced14ad4acfb827a1e33383d484f
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 14:53:43 2014 -0400

    initial commit
```

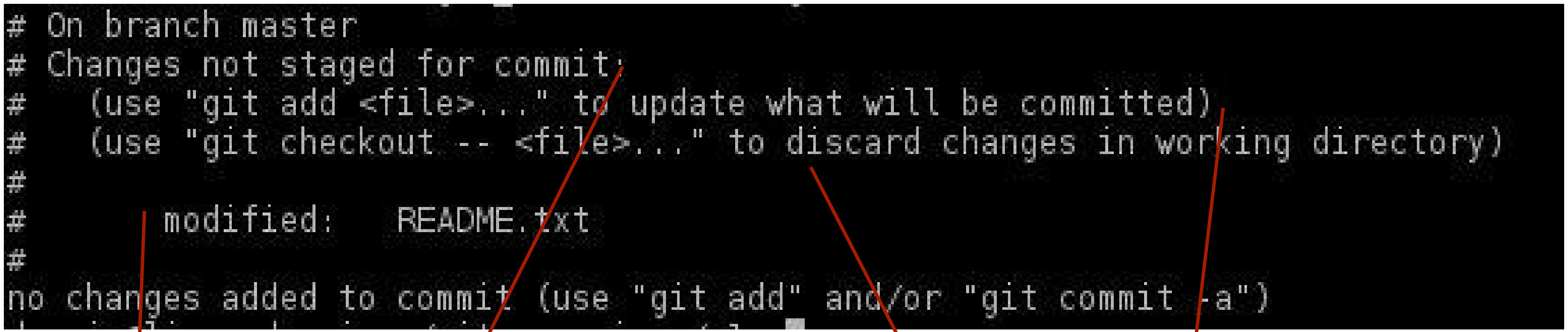
The message in our git commit command.

Author name and email set by *git config user. name "Dennis Jarecke"* and *git config user.email "jarecke@gmail.com"*

SHA1 of the commit

Git Exercise 1 (cont'd):

- Change the file to say "Hello universe!"
- *git status* . . . Finally we see something useful!



```
# 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:   README.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Git recognized that the file changed. Awesome!

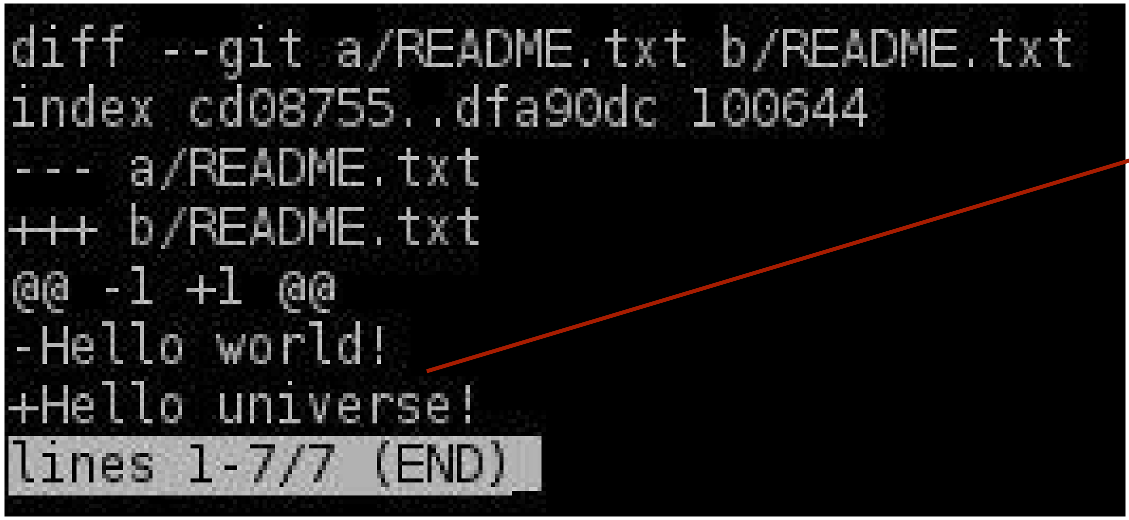
But, we haven't added it to the index with the *git add* command.

It tells you how to get rid of the changes. Try this one yourself!

And it reminds us to use the *git add* command to put it in the index to be committed.

Git Exercise 1 (cont'd):

- Pretend we didn't make the change to the file. How can we see what change was made? *git diff -- README.txt*

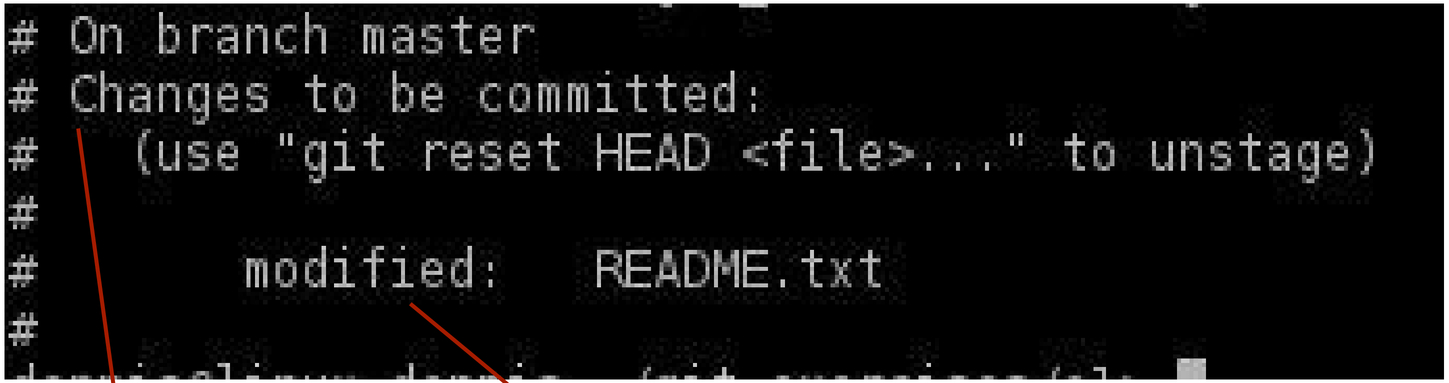


```
diff --git a/README.txt b/README.txt
index cd08755..dfa90dc 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-Hello world!
+Hello universe!
lines 1-7/7 (END)
```

Here it tells us we went from "Hello world!" to "Hello universe!"

Git Exercise 1 (cont'd):

- *git add README.txt* then *git status*



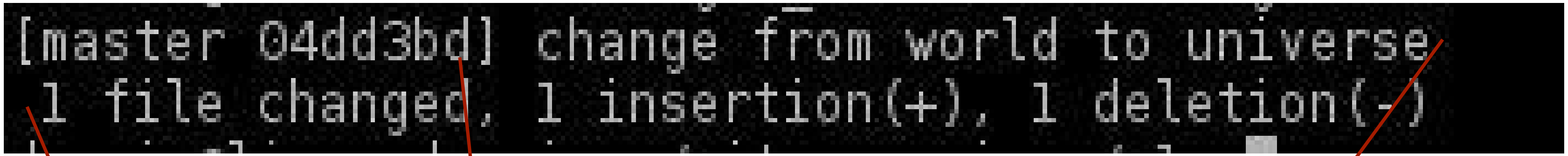
```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   README.txt
#
```

Again, it show the changes that need to be committed.

And now it is "modified" and not a "new file".

Git Exercise 1 (cont'd):

- *git commit -m "change from world to universe"*



```
[master 04dd3bd] change from world to universe
 1 file changed, 1 insertion(+), 1 deletion(-)
```

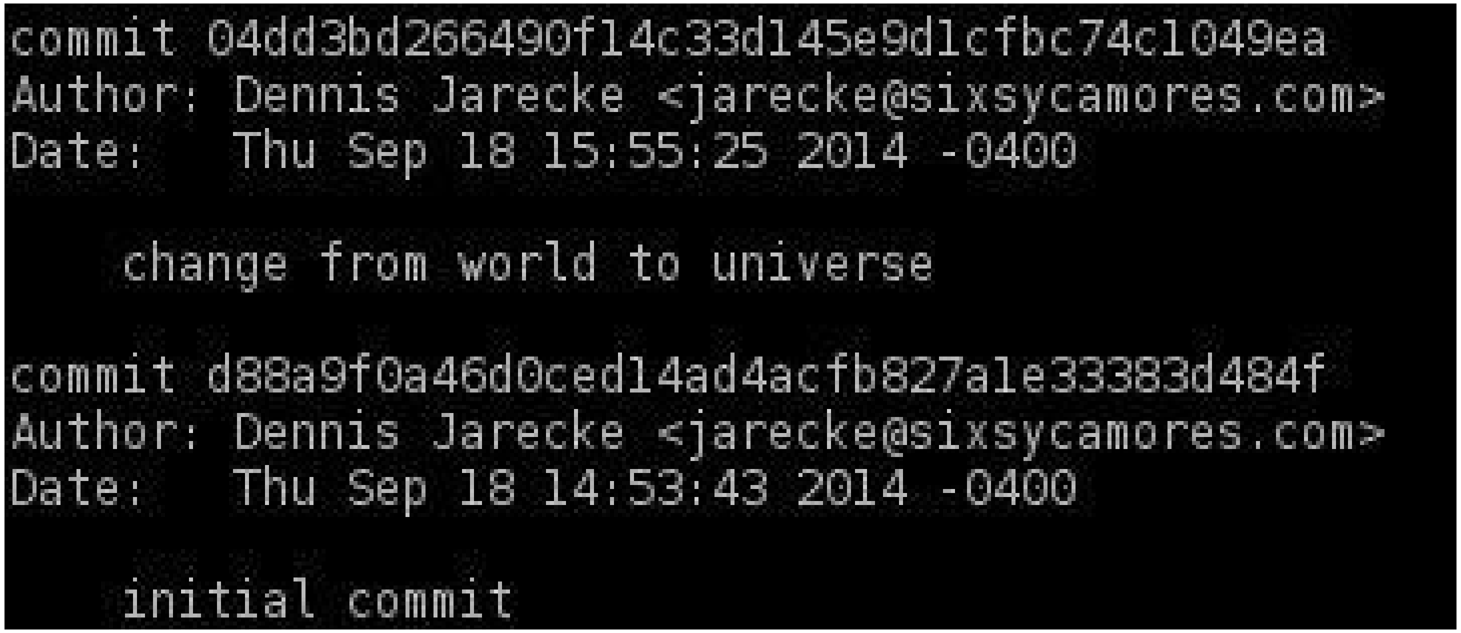
Summary of what was changed.

Shortened SHA1

Log message

Git Exercise 1 (cont'd):

- *git log*



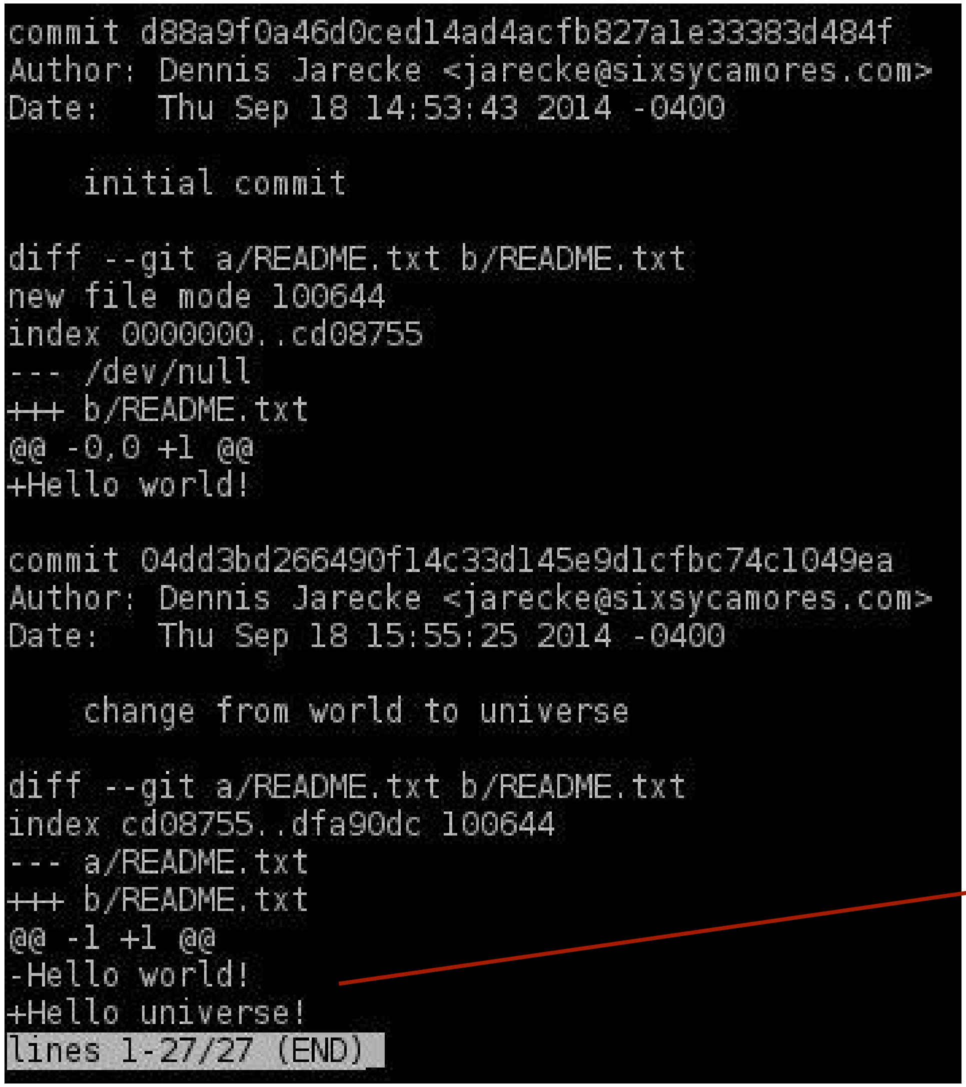
```
commit 04dd3bd266490f14c33d145e9d1cfbc74c1049ea
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 15:55:25 2014 -0400

    change from world to universe

commit d88a9f0a46d0ced14ad4acfb827a1e33383d484f
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 14:53:43 2014 -0400

    initial commit
```

Git log shows the SHA1s and commit messages - nothing new here, but you can see the progression of the changes.

Git Exercise 1 (cont'd):

- Now pretend we didn't make any of the commits. How can we see the changes between two commits?
- *git show d88a9f0a46d0ced14ad4acfb827a1e33383d484f 04dd3bd266490f14c33d145e9d1cfbc74c1049ea*
- See next slide

Git Exercise 1 (cont'd):



```
commit d88a9f0a46d0ced14ad4acfb827a1e33383d484f
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 14:53:43 2014 -0400

    initial commit

diff --git a/README.txt b/README.txt
new file mode 100644
index 0000000..cd08755
--- /dev/null
+++ b/README.txt
@@ -0,0 +1 @@
+Hello world!

commit 04dd3bd266490f14c33d145e9d1cfbc74c1049ea
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 15:55:25 2014 -0400

    change from world to universe

diff --git a/README.txt b/README.txt
index cd08755..dfa90dc 100644
--- a/README.txt
+++ b/README.txt
@@ -1 +1 @@
-Hello world!
+Hello universe!
lines 1-27/27 (END)
```

Here are the changes.

Git Exercise 1 (cont'd):

- Review
  - git init
  - git status
  - git add
  - git commit -m
  - git log
  - git config
  - git diff
  - git show
- Want more info?
  - git help init
  - git help status
  - git help add
  - . . .
  - You get the point of git help, right?

Hey, that's 17 commands!

Git Exercise 2:

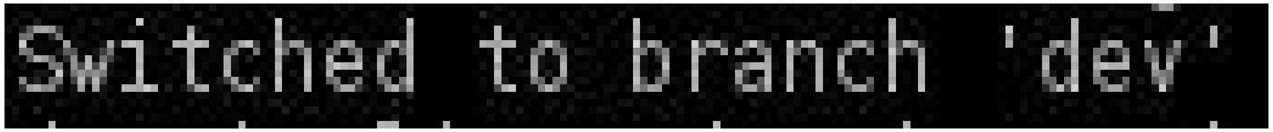
In the previous exercise we learned how to commit and view changes, but we only used one branch.

Let's view the master branch as our production code.

We will create a dev branch for development code and then merge that into production when our development is done.

Git Exercise 2 (cont'd):

- *git branch dev*
  - This creates a copy of the master branch.
  - There is no git output when you do this - crickets!
- *git status* shows we are still on the master branch so how do we get to the dev branch?
- *git checkout dev*



```
Switched to branch 'dev'
```

- *git diff master dev* will show no differences between the two branches

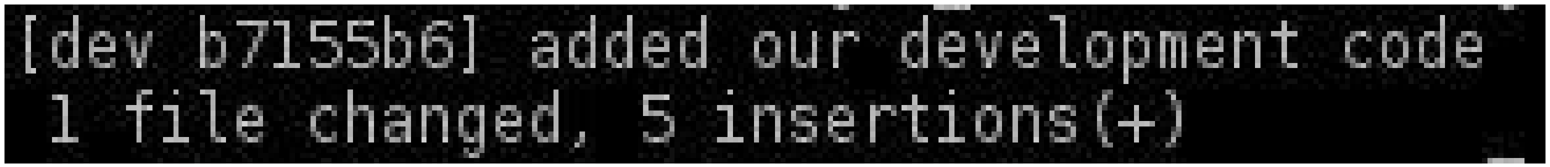
Git Exercise 2 (cont'd):

- Now let's modify README.txt to:

```
Hello universe!

/*
This is a bunch of development code that we are putting in.
What is your favorite language?  Mine is FORTRAN!  I'm
not kidding!
*/
```

Git Exercise 2 (cont'd):

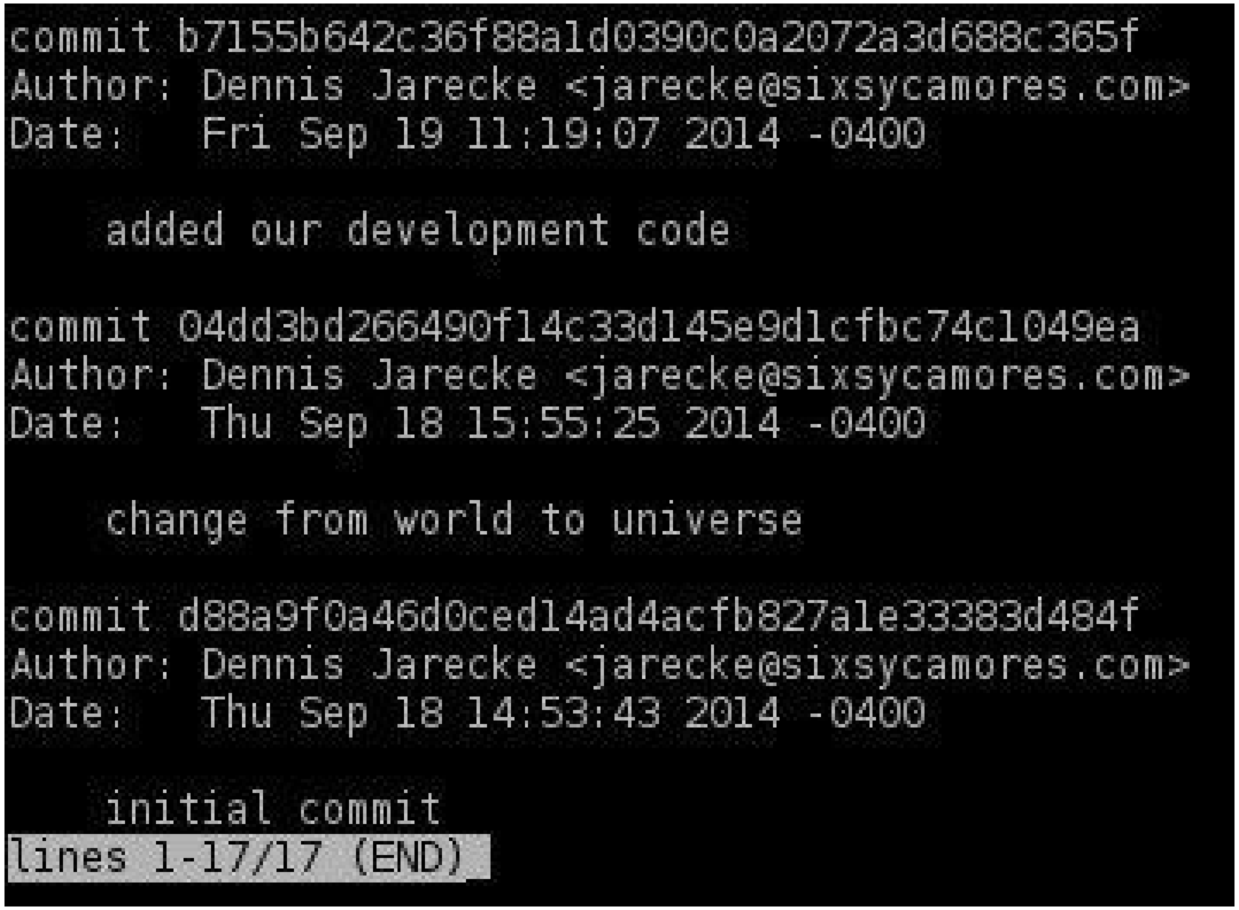
- Now because we already know the commands we will do the following:
  - *git status* will show that it is modified
  - *git diff* -- README.txt will show what has changed
  - *git add* will add it to the index
  - *git commit -m* "added our development code" will commit the changes



```
[dev b7155b6] added our development code
 1 file changed, 5 insertions(+)
```

Git Exercise 2 (cont'd):

❑ *git log* now shows the new commit!



```
commit b7155b642c36f88a1d0390c0a2072a3d688c365f
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Fri Sep 19 11:19:07 2014 -0400

    added our development code

commit 04dd3bd266490f14c33d145e9d1cfbc74c1049ea
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 15:55:25 2014 -0400

    change from world to universe

commit d88a9f0a46d0ced14ad4acfb827a1e33383d484f
Author: Dennis Jarecke <jarecke@sixsycamores.com>
Date:   Thu Sep 18 14:53:43 2014 -0400

    initial commit
lines 1-17/17 (END)
```

Git Exercise 2 (cont’d):

Now let’s pretend that we have tested the development code and we want to put it into production - i.e. the master branch. How do we do this?

We will merge the dev branch into the master branch. But before we do this we must check out master!

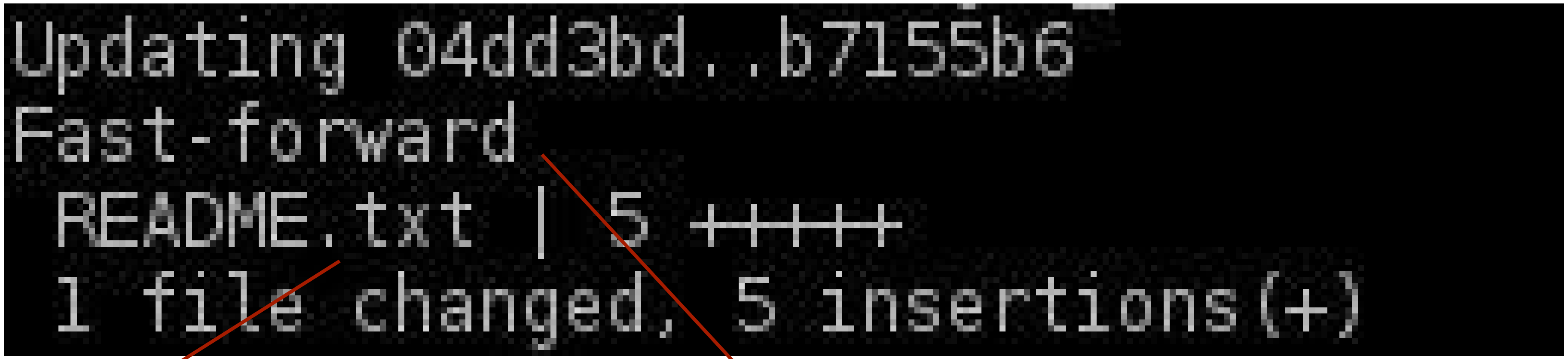
THIS IS REALLY IMPORTANT. YOU MUST ALWAYS CHECKOUT THE BRANCH YOU WANT THE CHANGES MERGED INTO!

Git Exercise 2 (cont'd):

- *git checkout master*
- (An aside - you can do things like *git log* to see that the development changes are not in and you can view the README.txt file and see that the development changes are not there.)
- Since we are moving code into production let's just verify what will be going in with *git diff master dev*. This will show all the details of the differences!
- But what if you have lots of file changes (we have only one here) and just want to see what files are changing? Use *git diff --stat master dev*.

Git Exercise 2 (cont'd):

- "Enough already! Just show me how to merge my changes!"
  - *git merge dev*
  - Verify with *git log* and looking at the file.



```
Updating 04dd3bd..b7155b6
Fast-forward
 README.txt | 5 +++++
 1 file changed, 5 insertions(+)
```

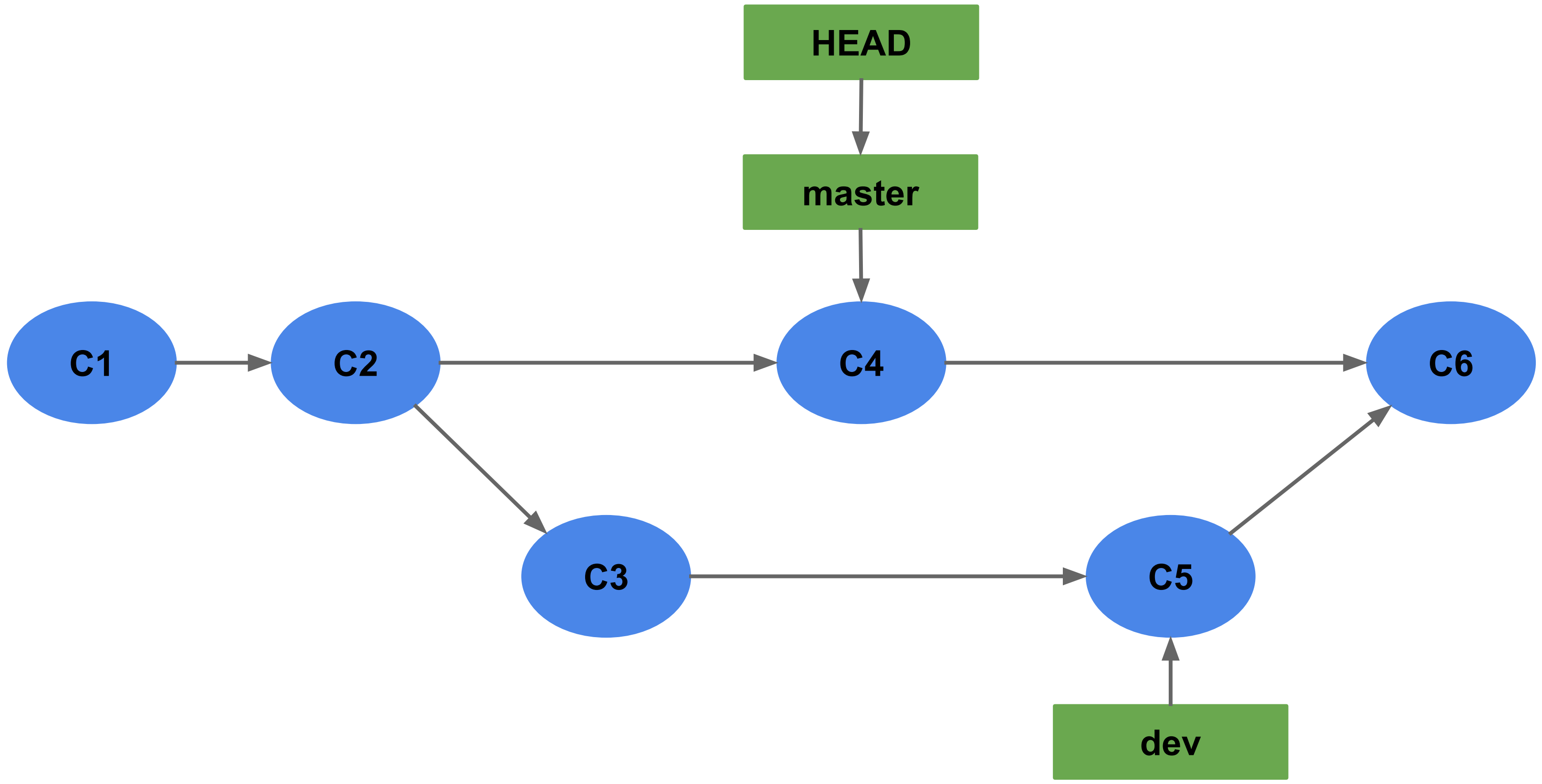
Shows us what has been changed.

What's this Fast-forward thing?

Fast-forward

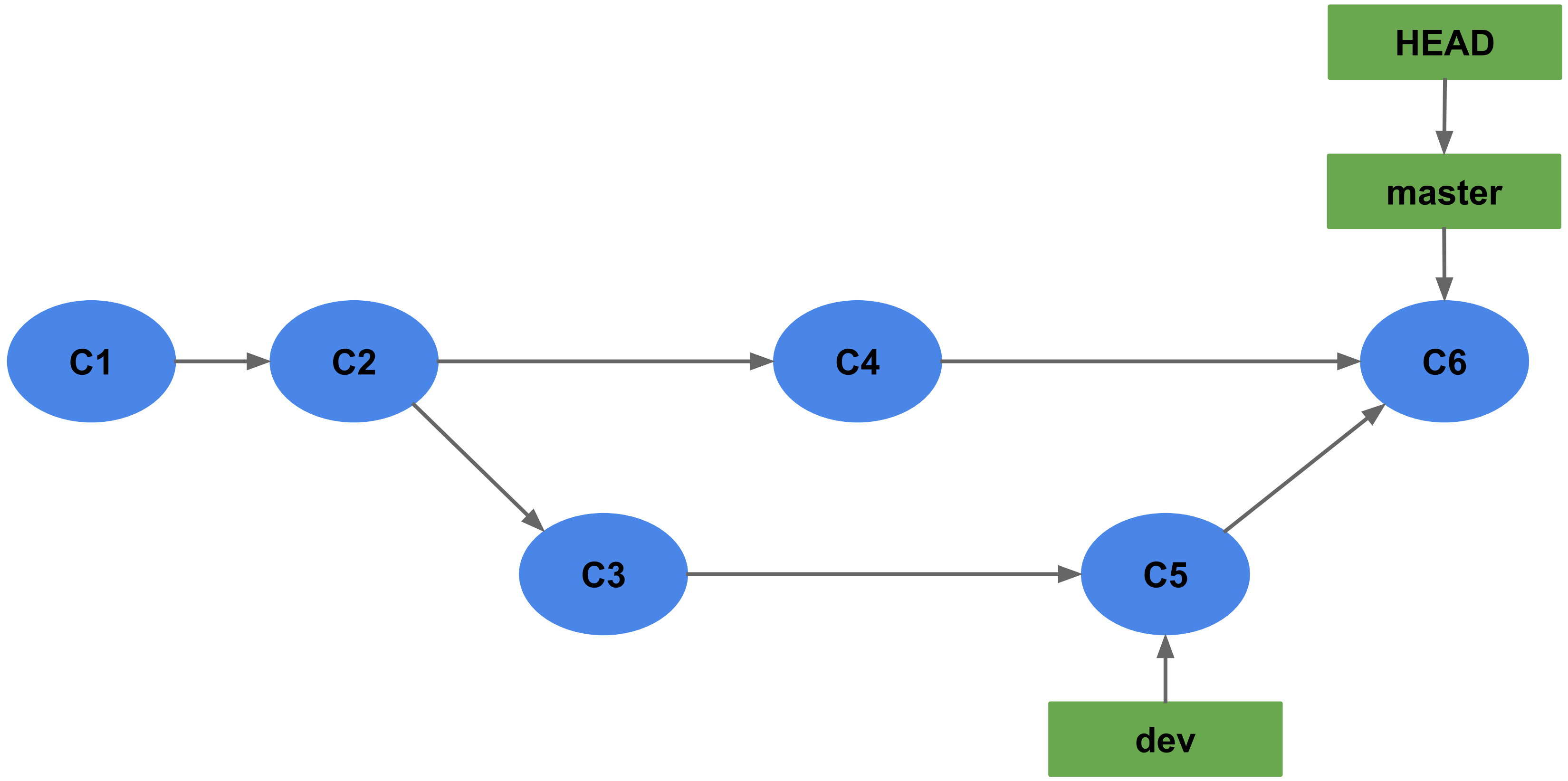
- How does git know what branch you are on and where you are at on the branch?
  - Think of the branch name as a pointer to the top commit of that branch.
  - HEAD is pointer/reference to the current branch name. HEAD points to the top commit of the current branch.
  - When you change branches you change where HEAD points to. Use *git show-ref* to see the commits each branch name points to.
- Think of a merge as a re-pointing of HEAD

Non-Fast-forward - Before git merge

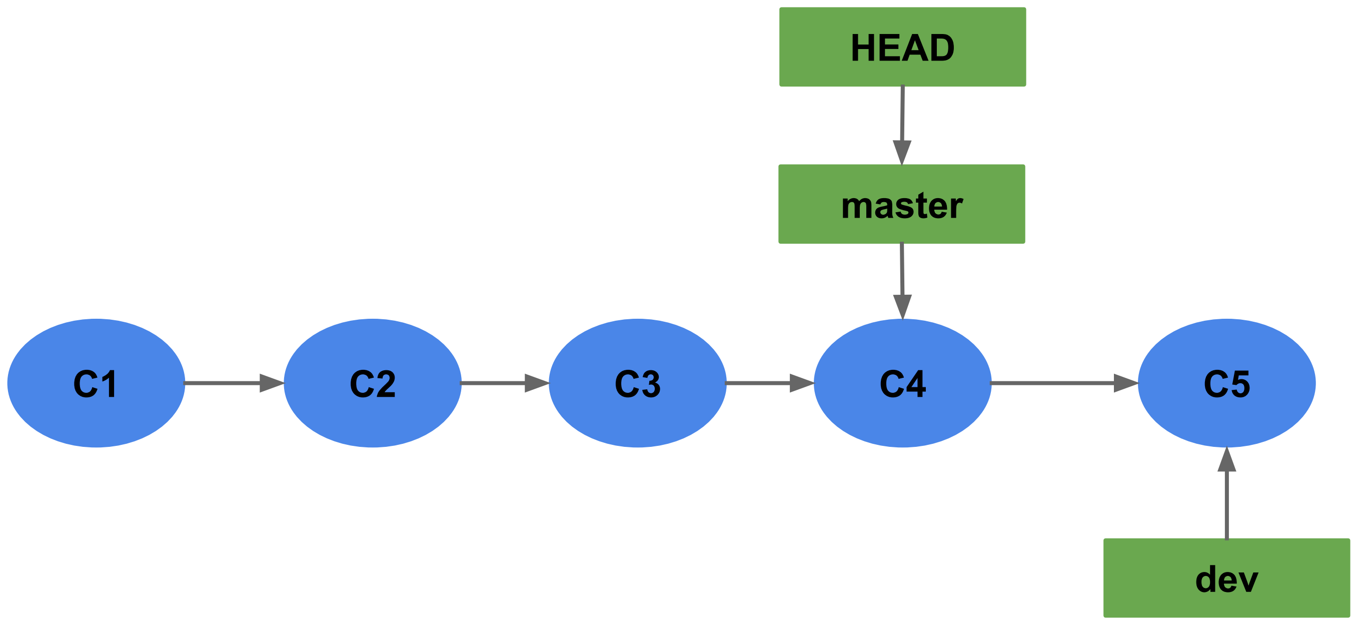


If I point HEAD to C5 then I lose C4 changes. Therefore I have to create a new commit (C6) that has both C4 and C5 changes and point HEAD to it. Therefore I can't fast-forward.

Non-Fast-forward - After git merge

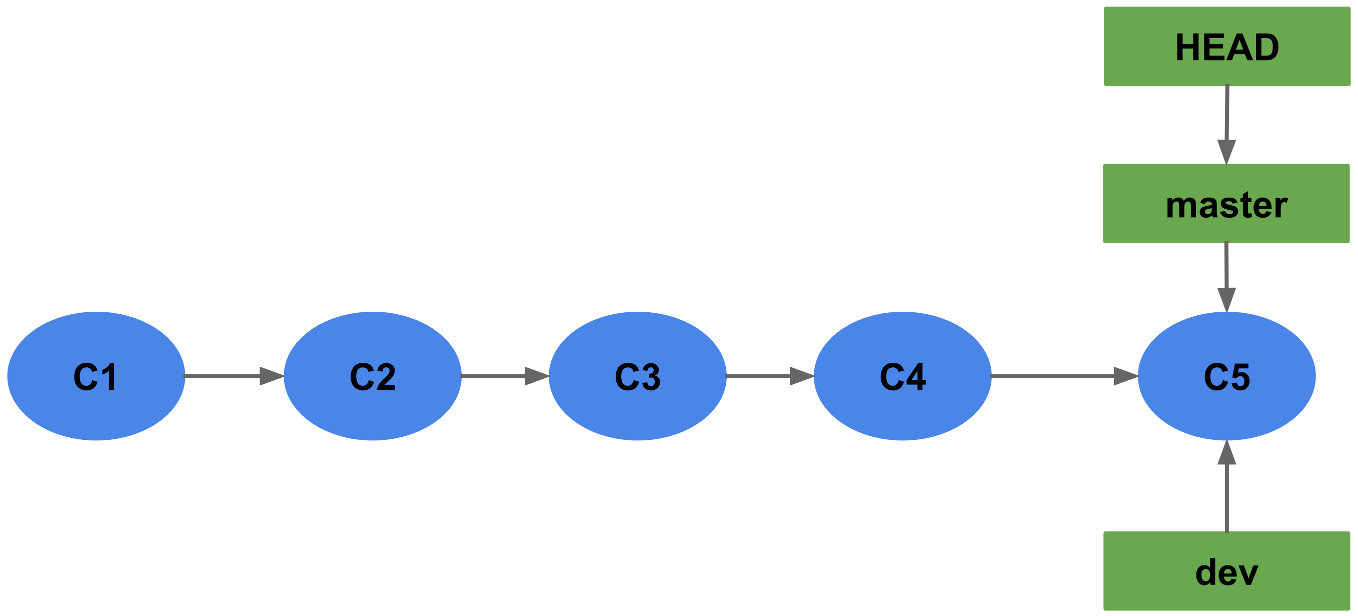


Fast-forward - Before git merge



If I point HEAD to C5 then I still get the change in C4. I do not have to create a new commit to do it. Therefore I CAN fast-forward.

Fast-forward - After git merge



What is Git Hub

An online set of git repositories used for both open source projects and for companies.

- Accounts are FREE
- Public repositories are FREE
- “Micro” account is $7/month and gives you 5 private repositories.

Git Exercise 3

A *remote* is a reference to a branch in another repository.

Think of it as two things (this may or may not be accurate but is how I think of it)

- A commit (i.e. SHA1) just like a branch name points to a local commit.
- A method for reaching the repository
  - A Linux path (if it's on the same computer.)
  - file (Yes, there is a difference from above.)
  - ssh
  - https
  - git (Some sort of git way of connecting repositories. Don't ask because I don't know.)

Git Exercise 3 (cont’d):

Let’s clone (yes, this is official git speak) a git hub repository with the following command:

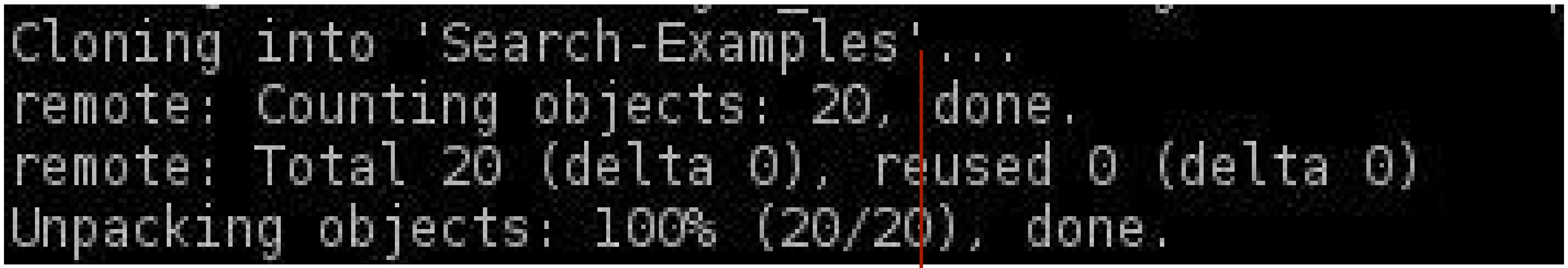
- *git clone https://github.com/tetmo113/Search-Examples*

- This is one of my public repositories - you can select any public or private repository you have access to.

- Even if you are not a collaborator you can still clone a repository! You just can’t push back into it!

Git Exercise 3 (cont'd):

- git clone https://github.com/tetmo113/Search-Examples

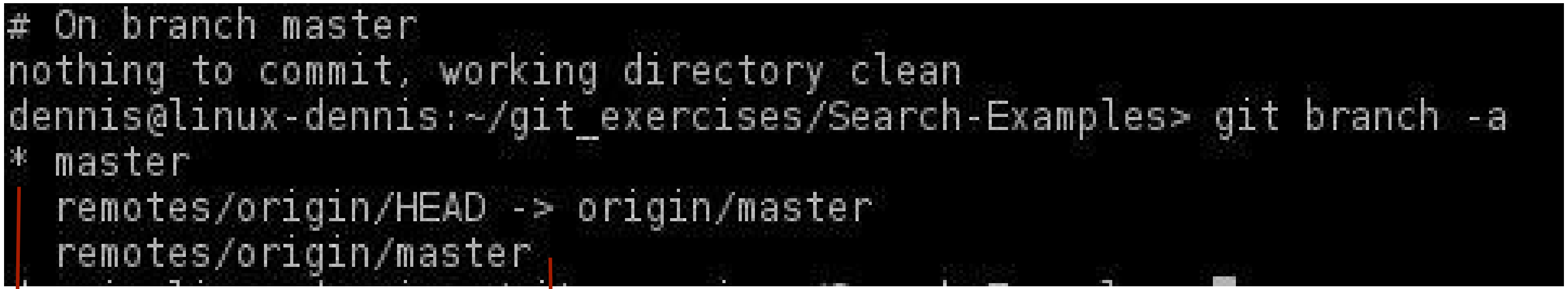


```
Cloning into 'Search-Examples'...
remote: Counting objects: 20, done.
remote: Total 20 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (20/20), done.
```

This will default to putting the repository into a folder named after the Git Hub repository. You can always do something like
git clone https://github.com/tetmo113/Search-Examples myrepositoryname
to put the repository into a directory named myrepositoryname.

Git Exercise 3 (cont'd):

- *git branch -a*



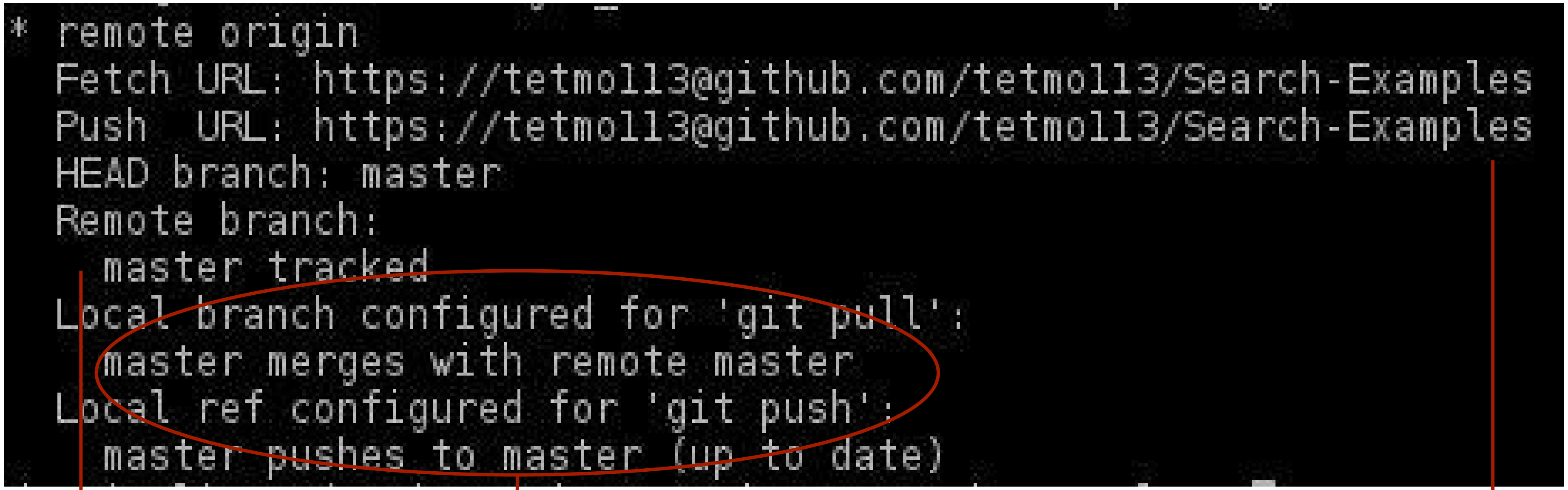
```
# On branch master
nothing to commit, working directory clean
dennis@linux-dennis:~/git_exercises/Search-Examples> git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/master
```

It automatically creates a local master branch.

Remember a remote branch points to a commit (branch name) on a remote machine. You can have many remotes. Just like a beginning default branch is called master, we call the default remote "origin". The remote named origin was automatically configured to point to https://github.com/tetmo113/Search-Examples when we di the *git clone* command.

Git Exercise 3 (cont'd):

- *git remote show* will list your remotes
- *git remote show origin* will show details about the origin remote

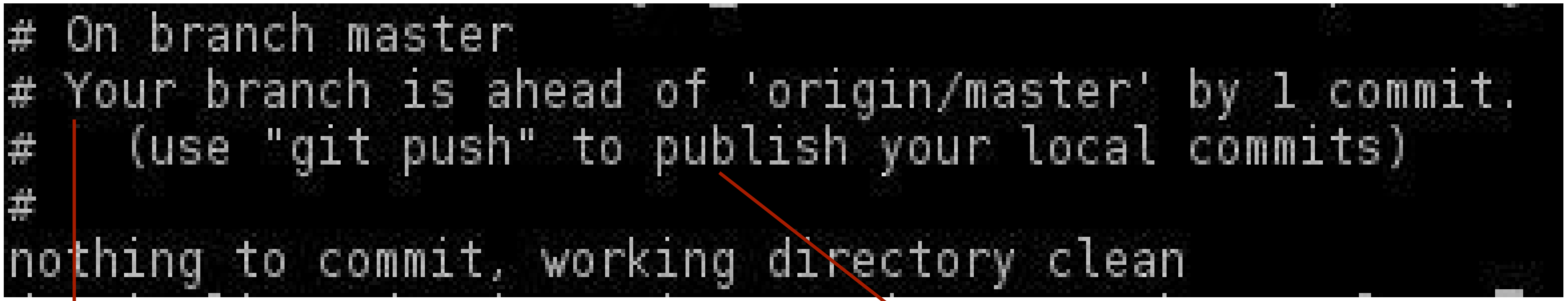


```
* remote origin
  Fetch URL: https://tetmoll13@github.com/tetmoll13/Search-Examples
  Push  URL: https://tetmoll13@github.com/tetmoll13/Search-Examples
  HEAD branch: master
  Remote branch:
    master tracked
  Local branch configured for 'git pull':
    master merges with remote master
  Local ref configured for 'git push':
    master pushes to master (up to date)
```

Local master tracks origin/master
Lots of good info. We'll talk about pull and push later.
URL of the remote

Git Exercise 3 (cont'd):

- Change README.txt, *git add*, *git commit*, and then *git status*. (Just doing what we learned in Exercise 1.)



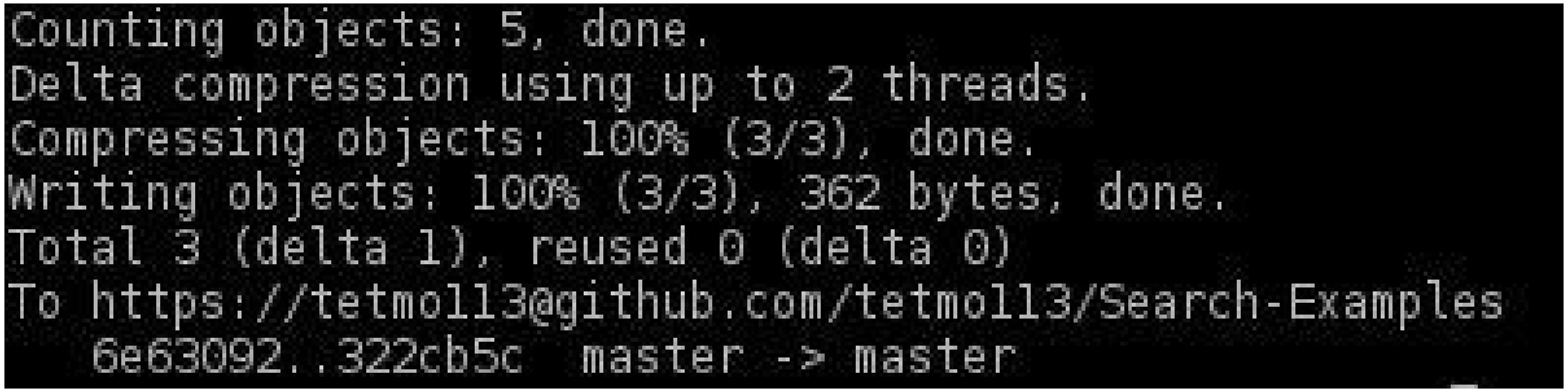
```
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

Because it tracks origin/master it knows that it is one commit ahead of it. Wow! Now we have the makings of a decentralized source control system. This is how everyone on the planet knows Linus has updated something. Actually in that case it will say our local master branch is behind origin/master by so many commits, but you git (pun intended) the point.

Another hint on what to do to "push" your changes to the origin server/remote. Gee, I wonder what we are going to do next.

Git Exercise 3 (cont’d):

- *git push origin master* (and type in your credentials) will move your local changes to the master branch on the remote repository.



```
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 362 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://tetmol13@github.com/tetmol13/Search-Examples
   6e63092..322cb5c  master -> master
```

Probably a lot that could be said here, but I think you get the point. We have pushed the changes to the remote repository. Now let’s head on over to github.com and see if in fact the changes made it there.

git pull and git fetch

- *git fetch --all* will fetch all remote branches and put them in things like origin/master, origin/dev, etc. This is safe in that it does not merge anything into local branches.
- *git pull* will fetch AND merge into your local tracking branches. You may or may not want this. So *git pull origin master* will update origin/master and then (behind the scenes) do *git merge origin/master*.

Tidbits

- You can do wicked things by pointing HEAD to different commits. State and Federal laws may apply.
- *git branch --track bar origin/foo* to make a new (local) branch bar and have it track origin/foo
- *git branch --set-upstream bar origin/foo* to make existing (local) branch bar track origin/foo
- *git add -u* will do a *git add* on everything that is tracked and has been changed. Very convenient if you update a lot of files - like in a Drupal update.

Tidbits (cont'd)

- *git add -A* will do *git add* on EVERYTHING including those files that have not been added to the repository.
- The .gitignore file is where you place files that you want git to ignore. *cough* settings.php *cough*
- *git checkout <commit hash> <filename>* will revert a single file back to a specific commit.
- *git log --full-history --all <filename>* to see full commit history of a single file.
- *git merge --no-ff* forces a new commit even if you can do a fast-forward

Tidbits (cont'd)

- *git rm --cached <file>* to remove a file from the repository but not the directory.
- *git reset --hard origin/master* force the current branch to be the same as origin/master. I use this all the time on a non-tracking generic dev branch.
- *git ls-files* to find a file in the repository
- *git log --graph --oneline --all* to see a graph of all commits over time - cool. Others may find this useful but I haven't.
- *git ls-remote* like *git show-ref* but for remote branches.

Tidbits (cont'd)

- Git GUIs
  - http://www.syntevo.com/smartgit/
  - gitk
- *git show <commit>* to see all the changes from a single commit. Can also use *git show --stat <commit>* to see only the files that were changed.
- *git branch --contains <commit>* to find branches that have this commit
- *git cherry-pick <commit>* Normally merging a commit will merge all the commits before it as well. This will mrege in ONLY the changes from this commit. Very useful, but be careful.

Links

- git-scm.com
- en.wikipedia.org/wiki/Git_(software)
- github.com
- gitref.org
- eagain.net/articles/git-for-computer-scientists