Часть 24

**Goals**

* Learn how to create a local branch in a repository

It’s time to do a major rewrite of the hello world functionality. Since this might take awhile, you’ll want to put these changes into a separate branch to isolate them from changes in master.

**Create a Branch *01***

Let’s call our new branch ‘greet’.

**Execute:**

git checkout -b greet

git status

**NOTE:** git checkout -b <branchname> is a shortcut for git branch <branchname> followed by a git checkout <branchname>.

Notice that the git status command reports that you are on the ‘greet’ branch.

**Changes for Greet: Add a Greeter class. *02***

**File: *lib/greeter.rb***

class Greeter

def initialize(who)

@who = who

end

def greet

"Hello, #{@who}"

end

end

**Execute:**

git add lib/greeter.rb

git commit -m "Added greeter class"

**Changes for Greet: Modify the main program *03***

Update the hello.rb file to use greeter

**File: *lib/hello.rb***

require 'greeter'

# Default is World

name = ARGV.first || "World"

greeter = Greeter.new(name)

puts greeter.greet

**Execute:**

git add lib/hello.rb

git commit -m "Hello uses Greeter"

**Changes for Greet: Update the Rakefile *04***

Update the Rakefile to use an external ruby process

**File: *Rakefile***

#!/usr/bin/ruby -wKU

task :default => :run

task :run do

ruby '-Ilib', 'lib/hello.rb'

end

**Execute:**

git add Rakefile

git commit -m "Updated Rakefile"

**Up Next *05***

We now have a new branch called **greet** with 3 new commits on it. Next we will learn how to navigate and switch between branches.

Часть 25

**Goals**

* Learn how to navigate between the branches of a repository

You now have two branches in your project:

**Execute:**

git hist --all

**Output:**

$ git hist --all

\* 28917a4 2013-04-13 | Updated Rakefile (HEAD, greet) [Jim Weirich]

\* 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

\* 96ee164 2013-04-13 | Added a Rakefile. (master) [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

**Switch to the Master Branch *01***

Just use the git checkout command to switch between branches.

**Execute:**

git checkout master

cat lib/hello.rb

**Output:**

$ git checkout master

Switched to branch 'master'

$ cat lib/hello.rb

# Default is World

# Author: Jim Weirich (jim@somewhere.com)

name = ARGV.first || "World"

puts "Hello, #{name}!"

You are now on the master branch. You can tell because the hello.rb file doesn’t use the Greeter class.

**Switch Back to the Greet Branch. *02***

**Execute:**

git checkout greet

cat lib/hello.rb

**Output:**

$ git checkout greet

Switched to branch 'greet'

$ cat lib/hello.rb

require 'greeter'

# Default is World

name = ARGV.first || "World"

greeter = Greeter.new(name)

puts greeter.greet

The contents of the lib/hello.rb confirms we are back on the **greet** branch.

Часть 26

**Goals**

* Learning how to deal with multiple branches with different (and possibly conflicting) changes.

While you were changing the greet branch, someone else decided to update the master branch. They added a README.

**Switch to the master branch. *01***

**Execute:**

git checkout master

**Create the README. *02***

**File: *README***

This is the Hello World example from the git tutorial.

**Commit the README to master. *03***

**Execute:**

git add README

git commit -m "Added README"

Часть 27

**Goals**

* Learn how to view diverging branches in a repository.

**View the Current Branches *01***

We now have two diverging branches in the repository. Use the following log command to view the branches and how they diverge.

**Execute:**

git hist --all

**Output:**

$ git hist --all

\* b59a8c2 2013-04-13 | Added README (HEAD, master) [Jim Weirich]

| \* 28917a4 2013-04-13 | Updated Rakefile (greet) [Jim Weirich]

| \* 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

| \* 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

|/

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

Here is our first chance to see the --graph option on git hist in action. Adding the --graph option to git log causes it to draw the commit tree using simple ASCII characters. We can see both branches (greet and master), and that the master branch is the current HEAD. The common ancestor to both branches is the “Added a Rakefile” branch.

The --all flag makes sure that we see all the branches. The default is to show only the current branch.

Часть 28

**Goals**

* Learn how to merge two diverging branches to bring the changes back into a single branch.

**Merge the branches *01***

Merging brings the changes in two branches together. Let’s go back to the greet branch and merge master onto greet.

**Execute:**

git checkout greet

git merge master

git hist --all

**Output:**

$ git checkout greet

Switched to branch 'greet'

$ git merge master

Merge made by recursive.

README | 1 +

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

create mode 100644 README

$ git hist --all

\* 844d1ed 2013-04-13 | Merge branch 'master' into greet (HEAD, greet) [Jim Weirich]

|\

| \* b59a8c2 2013-04-13 | Added README (master) [Jim Weirich]

\* | 28917a4 2013-04-13 | Updated Rakefile [Jim Weirich]

\* | 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* | 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

|/

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

By merging master into your greet branch periodically, you can pick up any changes to master and keep your changes in greet compatible with changes in the mainline.

However, it does produce ugly commit graphs. Later we will look at the option of rebasing rather than merging.

**Up Next *02***

But first, what if the changes in master conflict with the changes in greet?

Часть 29

**Goals**

* Create a conflicting change in the master branch.

**Switch back to master and create a conflict *01***

Switch back to the master branch and make this change:

**Execute:**

git checkout master

**File: *lib/hello.rb***

puts "What's your name"

my\_name = gets.strip

puts "Hello, #{my\_name}!"

**Execute:**

git add lib/hello.rb

git commit -m "Made interactive"

**View the Branches *02***

**Execute:**

git hist --all

**Output:**

$ git hist --all

\* 844d1ed 2013-04-13 | Merge branch 'master' into greet (greet) [Jim Weirich]

|\

\* | 28917a4 2013-04-13 | Updated Rakefile [Jim Weirich]

\* | 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* | 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

| | \* 05f32c0 2013-04-13 | Made interactive (HEAD, master) [Jim Weirich]

| |/

| \* b59a8c2 2013-04-13 | Added README [Jim Weirich]

|/

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

Master at commit “Added README” has been merged to the greet branch, but there is now an additional commit on master that has not been merged back to greet.

**Up Next *03***

The latest change in master conflicts with some existing changes in greet. Next we will resolve those changes.

Часть 30

**Goals**

* Learn how to handle conflicts during a merge

**Merge master to greet *01***

Now go back to the greet branch and try to merge the new master.

**Execute:**

git checkout greet

git merge master

**Output:**

$ git checkout greet

Switched to branch 'greet'

$ git merge master

Auto-merging lib/hello.rb

CONFLICT (content): Merge conflict in lib/hello.rb

Automatic merge failed; fix conflicts and then commit the result.

If you open lib/hello.rb, you will see:

**File: *lib/hello.rb***

<<<<<<< HEAD

require 'greeter'

# Default is World

name = ARGV.first || "World"

greeter = Greeter.new(name)

puts greeter.greet

=======

# Default is World

puts "What's your name"

my\_name = gets.strip

puts "Hello, #{my\_name}!"

>>>>>>> master

The first section is the version on the head of the current branch (greet). The second section is the version on the master branch.

**Fix the Conflict *02***

You need to manually resolve the conflict. Modify lib/hello.rb to be the following.

**File: *lib/hello.rb***

require 'greeter'

puts "What's your name"

my\_name = gets.strip

greeter = Greeter.new(my\_name)

puts greeter.greet

**Commit the Conflict Resolution *03***

**Execute:**

git add lib/hello.rb

git commit -m "Merged master fixed conflict."

**Output:**

$ git add lib/hello.rb

$ git commit -m "Merged master fixed conflict."

Recorded resolution for 'lib/hello.rb'.

[greet 25f0e8c] Merged master fixed conflict.

**Advanced Merging *04***

git doesn’t provide any graphical merge tools, but it will gladly work with any third party merge tool you wish to use. See <http://onestepback.org/index.cgi/Tech/Git/UsingP4MergeWithGit.red> for a description of using the Perforce merge tool with git.

Часть 31

**Goals**

* Learn the differences between rebasing and merging.

**Discussion**

Let’s explore the differences between merging and rebasing. In order to do so, we need to rewind the repository back in time before the first merge, and then redo the same steps, but using rebasing rather than merging.

We will make use the of the reset command to wind the branches back in time.

Часть 32

**Goals**

* Reset the greet branch to the point before the first merge.

**Reset the greet branch *01***

Let’s go back in time on the greet branch to the point *before* we merged master onto it. We can **reset** a branch to any commit we want. Essentially this is modifying the branch pointer to point to anywhere in the commit tree.

In this case we want to back greet up to the point prior to the merge with master. We need to find the last commit before the merge.

**Execute:**

git checkout greet

git hist

**Output:**

$ git checkout greet

Already on 'greet'

$ git hist

\* 25f0e8c 2013-04-13 | Merged master fixed conflict. (HEAD, greet) [Jim Weirich]

|\

| \* 05f32c0 2013-04-13 | Made interactive (master) [Jim Weirich]

\* | 844d1ed 2013-04-13 | Merge branch 'master' into greet [Jim Weirich]

|\ \

| |/

| \* b59a8c2 2013-04-13 | Added README [Jim Weirich]

\* | 28917a4 2013-04-13 | Updated Rakefile [Jim Weirich]

\* | 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* | 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

|/

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

That’s a bit hard to read, but looking at the data we see that the “Updated Rakefile” commit was the last commit on the greet branch before merging. Let’s reset the greet branch to that commit.

**Execute:**

git reset --hard <hash>

**Output:**

$ git reset --hard 28917a4

HEAD is now at 28917a4 Updated Rakefile

**Check the branch. *02***

Look at the log for the greet branch. We no longer have the merge commits in its history.

**Execute:**

git hist --all

**Output:**

$ git hist --all

\* 05f32c0 2013-04-13 | Made interactive (master) [Jim Weirich]

\* b59a8c2 2013-04-13 | Added README [Jim Weirich]

| \* 28917a4 2013-04-13 | Updated Rakefile (HEAD, greet) [Jim Weirich]

| \* 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

| \* 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

|/

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

Часть 33

**Goals**

* Reset the master branch to the point before the conflicting commit.

**Reset the master branch *01***

When we added the interactive mode to the master branch, we made a change that conflicted with changes in the greet branch. Let’s rewind the master branch to a point before the conflicting change. This allows us to demonstrate the rebase command without worrying about conflicts.

**Execute:**

git checkout master

git hist

**Output:**

$ git hist

\* 05f32c0 2013-04-13 | Made interactive (HEAD, master) [Jim Weirich]

\* b59a8c2 2013-04-13 | Added README [Jim Weirich]

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

The ‘Added README’ commit is the one directly before the conflicting interactive mode. We will reset the master branch to ‘Added README’ branch.

**Execute:**

git reset --hard <hash>

git hist --all

Review the log. It should look like the repository has been wound back in time to the point before we merged anything.

**Output:**

$ git hist --all

\* b59a8c2 2013-04-13 | Added README (HEAD, master) [Jim Weirich]

| \* 28917a4 2013-04-13 | Updated Rakefile (greet) [Jim Weirich]

| \* 4dac415 2013-04-13 | Hello uses Greeter [Jim Weirich]

| \* 39347b3 2013-04-13 | Added greeter class [Jim Weirich]

|/

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

Часть 34

**Goals**

* Use the rebase command rather than the merge command.

Ok, we are back in time before the first merge and we want to get the changes in master into our greet branch.

This time we will use the rebase command instead of the merge command to bring in the changes from the master branch.

**Execute:**

git checkout greet

git rebase master

git hist

**Output:**

$ go greet

Switched to branch 'greet'

$

$ git rebase master

First, rewinding head to replay your work on top of it...

Applying: added Greeter class

Applying: hello uses Greeter

Applying: updated Rakefile

$

$ git hist

\* 2fae0b2 2013-04-13 | Updated Rakefile (HEAD, greet) [Jim Weirich]

\* 1c23048 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* 62d7ce0 2013-04-13 | Added greeter class [Jim Weirich]

\* b59a8c2 2013-04-13 | Added README (master) [Jim Weirich]

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

**Merge VS Rebase *01***

The final result of the rebase is very similar to the merge. The greet branch now contains all of its changes, as well as all the changes from the master branch. However, the commit tree is quite different. The commit tree for the greet branch has been rewritten so that the master branch is a part of the commit history. This leaves the chain of commits linear and much easier to read.

**When to Rebase, When to Merge? *02***

Don’t use rebase …

1. If the branch is public and shared with others. Rewriting publicly shared branches will tend to screw up other members of the team.
2. When the *exact* history of the commit branch is important (since rebase rewrites the commit history).

Given the above guidelines, I tend to use rebase for short-lived, local branches and merge for branches in the public repository.

Часть 35

**Goals**

* We’ve kept our greet branch up to date with master (via rebase), now let’s merge the greet changes back into the master branch.

**Merge greet into master *01***

**Execute:**

git checkout master

git merge greet

**Output:**

$ git checkout master

Switched to branch 'master'

$

$ git merge greet

Updating b59a8c2..2fae0b2

Fast-forward

Rakefile | 2 +-

lib/greeter.rb | 8 ++++++++

lib/hello.rb | 6 ++++--

3 files changed, 13 insertions(+), 3 deletions(-)

create mode 100644 lib/greeter.rb

Because the head of master is a direct ancestor of the head of the greet branch, git is able to do a fast-forward merge. When fast-forwarding, the branch pointer is simply moved forward to point to the same commit as the greeter branch.

There will never be conflicts in a fast-forward merge.

**Review the logs *02***

**Execute:**

git hist

**Output:**

$ git hist

\* 2fae0b2 2013-04-13 | Updated Rakefile (HEAD, master, greet) [Jim Weirich]

\* 1c23048 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* 62d7ce0 2013-04-13 | Added greeter class [Jim Weirich]

\* b59a8c2 2013-04-13 | Added README [Jim Weirich]

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

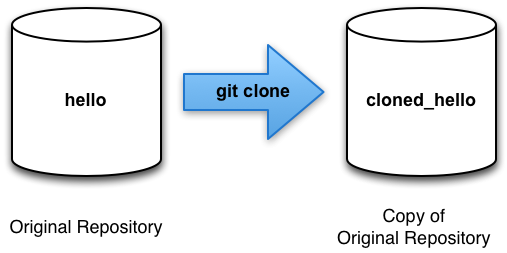
\* 9416416 2013-04-13 | First Commit [Jim Weirich]

The greet and master branches are now identical.

Часть 36

Up to this point we have been working with a single git repository. However, git excels at working with multiple repositories. These extra repositories may be stored locally, or may be accessed across a network connection.

In the next section we will create a new repository called “cloned\_hello”. We will show how to move changes from one repository to another, and how to handle conflicts when they arise from between two repositories.



For now, we will be working with local repositories (i.e. repositories stored on your local hard disk), however most of the things learned in this section will apply to multiple repositories whether they are stored locally or remotely over a network.

**NOTE:** We are going be making changes to both copies of our repositories. Make sure you pay attention to which repository you are in at each step of the following labs.

Часть 37

**Goals**

* Learn how to make copies of repositories.

**Go to the work directory *01***

Go to the working directory and make a clone of your hello repository.

**Execute:**

cd ..

pwd

ls

**NOTE: Now in the work directory.**

**Output:**

$ cd ..

$ pwd

/Users/jim/working/git/git\_immersion/auto

$ ls

hello

At this point you should be in your “work” directory. There should be a single repository here named “hello”.

**Create a clone of the hello repository *02***

Let’s make a clone of the repository.

**Execute:**

git clone hello cloned\_hello

ls

**Output:**

$ git clone hello cloned\_hello

Cloning into cloned\_hello...

done.

$ ls

cloned\_hello

hello

There should now be two repositories in your work directory: the original “hello” repository and the newly cloned “cloned\_hello” repository.

Часть 38

**Goals**

* Learn about branches on remote repositories.

**Look at the cloned repository *01***

Let’s take a look at the cloned repository.

**Execute:**

cd cloned\_hello

ls

**Output:**

$ cd cloned\_hello

$ ls

README

Rakefile

lib

You should see a list of all the files in the top level of the original repository (README, Rakefile and lib).

**Review the Repository History *02***

**Execute:**

git hist --all

**Output:**

$ git hist --all

\* 2fae0b2 2013-04-13 | Updated Rakefile (HEAD, origin/master, origin/greet, origin/HEAD, master) [Jim Weirich]

\* 1c23048 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* 62d7ce0 2013-04-13 | Added greeter class [Jim Weirich]

\* b59a8c2 2013-04-13 | Added README [Jim Weirich]

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

You should now see a list of the all the commits in the new repository, and it should (more or less) match the history of commits in the original repository. The only difference should be in the names of the branches.

**Remote branches *03***

You should see a **master** branch (along with **HEAD**) in the history list. But you will also have number of strangely named branches (**origin/master**, **origin/greet** and **origin/HEAD**). We’ll talk about them in a bit.

Часть 39

**Goals**

* Learn about naming remote repositories.

**Execute:**

git remote

**Output:**

$ git remote

origin

We see that the cloned repository knows about a remote repository named origin. Let’s see if we can get more information about origin:

**Execute:**

git remote show origin

**Output:**

$ git remote show origin

\* remote origin

Fetch URL: /Users/jim/working/git/git\_immersion/auto/hello

Push URL: /Users/jim/working/git/git\_immersion/auto/hello

HEAD branch (remote HEAD is ambiguous, may be one of the following):

greet

master

Remote branches:

greet tracked

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)

Now we see that the remote repository “origin” is simply the original **hello** repository. Remote repositories typically live on a separate machine, possibly a centralized server. As we can see here, however, they can just as well point to a repository on the same machine. There is nothing particularly special about the name “origin”, however the convention is to use the name “origin” for the primary centralized repository (if there is one).

Часть 40

**Goals**

* Learn about local VS remote branches

Let’s look at the branches available in our cloned repository.

**Execute:**

git branch

**Output:**

$ git branch

\* master

That’s it, only the master branch is listed. Where is the greet branch? The **git** **branch** command only lists the local branches by default.

**List Remote Branches *01***

Try this to see all the branches:

**Execute:**

git branch -a

**Output:**

$ git branch -a

\* master

remotes/origin/HEAD -> origin/master

remotes/origin/greet

remotes/origin/master

Git has all the commits from the original repository, but branches in the remote repository are not treated as local branches here. If we want our own **greet** branch, we need to create it ourselves. We will see how to do that in a minute.

Часть 41

**Goals**

* Make some changes to the original repository so we can try to pull the changes

**Make a change in the original hello repository *01***

**Execute:**

cd ../hello

# (You should be in the original hello repository now)

**NOTE: Now in the *hello* repo**

Make the following changes to README:

**File: *README***

This is the Hello World example from the git tutorial.

(changed in original)

Now add and commit this change

**Execute:**

git add README

git commit -m "Changed README in original repo"

**Up Next *02***

The original repository now has later changes that are not in the cloned version. Next we will pull those changes across to the cloned repository.

Часть 42

**Goals**

* Learn how to pull changes from a remote repository.

**Execute:**

cd ../cloned\_hello

git fetch

git hist --all

**NOTE: Now in the *cloned\_hello* repo**

**Output:**

$ git fetch

From /Users/jim/working/git/git\_immersion/auto/hello

2fae0b2..2e4c559 master -> origin/master

$ git hist --all

\* 2e4c559 2013-04-13 | Changed README in original repo (origin/master, origin/HEAD) [Jim Weirich]

\* 2fae0b2 2013-04-13 | Updated Rakefile (HEAD, origin/greet, master) [Jim Weirich]

\* 1c23048 2013-04-13 | Hello uses Greeter [Jim Weirich]

\* 62d7ce0 2013-04-13 | Added greeter class [Jim Weirich]

\* b59a8c2 2013-04-13 | Added README [Jim Weirich]

\* 96ee164 2013-04-13 | Added a Rakefile. [Jim Weirich]

\* 0f36766 2013-04-13 | Moved hello.rb to lib [Jim Weirich]

\* eb30103 2013-04-13 | Add an author/email comment [Jim Weirich]

\* 1f7ec5e 2013-04-13 | Added a comment (v1) [Jim Weirich]

\* 582495a 2013-04-13 | Added a default value (v1-beta) [Jim Weirich]

\* 323e28d 2013-04-13 | Using ARGV [Jim Weirich]

\* 9416416 2013-04-13 | First Commit [Jim Weirich]

At this point the repository has all the commits from the original repository, but they are not integrated into the the cloned repository’s local branches.

Find the “Changed README in original repo” commit in the history above. Notice that the commit includes “origin/master” and “origin/HEAD”.

Now look at the “Updated Rakefile” commit. You will see that it the local master branch points to this commit, not to the new commit that we just fetched.

The upshot of this is that the “git fetch” command will fetch new commits from the remote repository, but it will not merge these commits into the local branches.

**Check the README *01***

We can demonstrate that the cloned README is unchanged.

**Execute:**

cat README

**Output:**

$ cat README

This is the Hello World example from the git tutorial.

See, no changes.

Часть 43

**Goals**

* Learn to get the pulled changes into the current branch and working directory.

**Merge the fetched changes into local master *01***

**Execute:**

git merge origin/master

**Output:**

$ git merge origin/master

Updating 2fae0b2..2e4c559

Fast-forward

README | 1 +

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

**Check the README again *02***

We should see the changes now.

**Execute:**

cat README

**Output:**

$ cat README

This is the Hello World example from the git tutorial.

(changed in original)

There are the changes. Even though “git fetch” does not merge the changes, we can still manually merge the changes from the remote repository.

**Up Next *03***

Next let’s take a look at combining the fetch & merge process into a single command.

Часть 44

**Goals**

* Learn that git pull is equivalent to a git fetch followed by a git merge.

**Discussion**

We’re not going to go through the process of creating another change and pulling it again, but we do want you to know that doing:

git pull

is indeed equivalent to the two steps:

git fetch

git merge origin/master

Часть 45

**Goals**

* Learn how to add a local branch that tracks a remote branch.

The branches starting with remotes/origin are branches from the original repo. Notice that you don’t have a branch called greet anymore, but it knows that the original repo had a greet branch.

**Add a local branch that tracks a remote branch. *01***

**Execute:**

git branch --track greet origin/greet

git branch -a

git hist --max-count=2

**Output:**

$ git branch --track greet origin/greet

Branch greet set up to track remote branch greet from origin.

$ git branch -a

greet

\* master

remotes/origin/HEAD -> origin/master

remotes/origin/greet

remotes/origin/master

$ git hist --max-count=2

\* 2e4c559 2013-04-13 | Changed README in original repo (HEAD, origin/master, origin/HEAD, master) [Jim Weirich]

\* 2fae0b2 2013-04-13 | Updated Rakefile (origin/greet, greet) [Jim Weirich]

We can now see the greet branch in the branch list and in the log.

Часть 46

**Goals**

* Learn how to create bare repositories.

Bare repositories (without working directories) are usually used for sharing.

**Create a bare repository. *01***

**Execute:**

cd ..

git clone --bare hello hello.git

ls hello.git

**NOTE: Now in the work directory**

**Output:**

$ git clone --bare hello hello.git

Cloning into bare repository hello.git...

done.

$ ls hello.git

HEAD

config

description

hooks

info

objects

packed-refs

refs

The convention is that repositories ending in ‘.git’ are bare repositories. We can see that there is no working directory in the hello.git repo. Essentially it is nothing but the .git directory of a non-bare repo.

Часть 47

**Goals**

* Add the bare repository as a remote to our original repository.

Let’s add the hello.git repo to our original repo.

**Execute:**

cd hello

git remote add shared ../hello.git

**NOTE: Now in the hello repository.**

Часть 48

**Goals**

* Learn how out to push a change to a remote repository.

Since bare repositories are usually shared on some sort of network server, it is usually difficult to cd into the repo and pull changes. So we need to push our changes into other repositories.

Let’s start by creating a change to be pushed. Edit the README and commit it

**File: *README***

This is the Hello World example from the git tutorial.

(Changed in the original and pushed to shared)

**Execute:**

git checkout master

git add README

git commit -m "Added shared comment to readme"

Now push the change to the shared repo.

**Execute:**

git push shared master

*shared* is the name of the repository receiving the changes we are pushing. (Remember, we added it as a remote in the previous lab.)

**Output:**

$ git push shared master

To ../hello.git

2e4c559..3923dd5 master -> master

**NOTE:** We had to explicitly name the branch master that was receiving the push. It is possible to set it up automatically, but I *never* remember the commands to do that. Check out the “Git Remote Branch” gem for easy management of remote branches.

Часть 49

**Goals**

* Learn how to pull changes from a shared repository.

Quick hop over to the clone repository and let’s pull down the changes just pushed to the shared repo.

**Execute:**

cd ../cloned\_hello

**NOTE: Now in the *cloned\_hello* repo.**

Continue with…

**Execute:**

git remote add shared ../hello.git

git branch --track shared master

git pull shared master

cat README

Часть 50

**Goals**

* Learn how to setup git server for sharing repositories.

There are many ways to share git repositories over the network. Here is a quick and dirty way.

**Start up the git server *01***

**Execute:**

# (From the work directory)

git daemon --verbose --export-all --base-path=.

Now, in a separate terminal window, go to your work directory

**Execute:**

# (From the work directory)

git clone git://localhost/hello.git network\_hello

cd network\_hello

ls

You should see a copy of hello project.

**Pushing to the Git Daemon *02***

If you want to push to the git daemon repository, add --enable=receive-pack to the git daemon command. Be careful because there is no authentication on this server, anyone could push to your repository.

Часть 51

**Goals**

* Learn to share repos across WIFI.

See if your neighbor is running the git daemon. Exchange IP addresses and see if you can pull from each other’s repositories.

**NOTE:** The gitjour gem is really useful in sharing ad-hoc repositories.

Часть 52

Here are some topics you might want to research on your own:

* Reverting Committed Changes
* Cross OS Line Endings
* Remote Servers
* Protocols
* SSH Setup
* Remote Branch Management
* Finding Buggy Commits (git bisect)
* Workflows
* Non-command line tools (gitx, gitk, magit)
* Working with GitHub