Git Tutorial

This tutorial is based on a subset of this online tutorial: `http://gitimmersion.com/index.html`.

1 Setup

Execute:

```
git config --global user.name "NAME"
git config --global user.email "EMAIL"
git config --global core.editor "EDITOR"
```

When you execute the commands, substitute `NAME` with your name, `EMAIL` with your Kutztown University email address, and `EDITOR` with your preferred text editor. Note that the double quotes need to be included. If you do not have a preferred text editor, then I suggest setting the editor to `"nano"`.

2 Create a project

Create an empty directory named "`work/tutorial`" and then change into that directory.

Execute:

```
mkdir -p work/tutorial
cd work/tutorial
```

Use your preferred text editor to create a new file named `todo.md` with the following content:

```
* Do homework
```

You are now in a directory that contains a single file. The `git init` command creates a repository in the current working directory.

Execute:

```
git init
```

Output:

```
Initialized empty Git repository in .../work/tutorial/.git/
```

Add the `todo.md` file to the repository.

Execute:

```
git add todo.md
git commit -m "Initial Commit"
```

Output:

```
[master (root-commit) fe1c3e4] Initial Commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 todo.md
```

3 Checking the status

The `git status` command checks the current status of the repository.

Execute:

```
git status
```

Output:

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

The status command reports that there is nothing to commit. This means that the repository has the current state of the working directory, that is, there are no outstanding changes to record.

4 Making Changes

Change the `todo.md` file to have the following content:

```
* Do homework
* Buy groceries
```

Check the status of the working directory.

Execute:

```
git status
```

Output:

```
+ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   todo.md
#
```

This message indicates that git detected that the `todo.md` file has been modified, but the changes are not yet in the repository. Also, note that the status message gives you hints about what you need to do next. Namely, if you want to add the changes to the repository, then use the `git add` command. Otherwise, the `git checkout` command can be used to discard the changes.

5 Staging Changes

Execute the `git add` command to stage the changes, then check the status.

Execute:

```
git add todo.md
git status
```

Output:

```
+ git add todo.md
+ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   todo.md
#
```

The change to the `todo.md` file has been staged. This means that git knows about the change, but the change has not been permanently recorded in the repository yet. The next `git commit` operation will include the staged changes.

If you decide that you do not want to commit the change, the `git status` message indicates that the `git reset` command can be used to unstage that change.

6 Committing Changes

The `git commit` command used in Section 2 included the `-m` flag followed by a message in double quotes. If the `-m` flag is omitted, then git will open a text editor of your choice (this was setup in Section 1). The editor is chosen from the following list (in priority order):

- the `GIT_EDITOR` environment variable
- the `core.editor` configuration setting
- the `VISUAL` environment variable
- the `EDITOR` environment variable

Run the `git commit` command without the `-m` flag

Execute:

```
git commit
```

A text editor will open with the following

```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch master
# Changes to be committed:
#       modified:   todo.md
#
```

On the first line, enter the text: `Add second todo item`. Then save the file and exit the text editor.

Output:

```
+ git commit
[master 2ef381d] Add second todo item
 1 files changed, 1 insertions(+), 0 deletions(-)
```

Then check the status again.

```
git status
```

Output:

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

The working directory is clean and ready for you to continue.

7 Changes, not Files

Most version control systems work with files; when a file is added, the system tracks changes to that file from that point on.

Git focuses on the changes to a file rather than the file itself. The `git add` command does not add the file to the repository. Instead, the `git add` command tells git to make a note of the current state of the file to be committed later. Here we will explore that difference.

Add one more item to our list in the `todo.md` file:

File: `todo.md`

```
* Do homework
* Buy groceries
* Study for exam
```

Add the change to git's staging area.

Execute:

```
git add todo.md
```

Make another edit to the `todo.md` file.

File: `todo.md`

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
```

Check the status

```
git status
```

Output:

```
+ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   todo.md
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   todo.md
#
```

Note that `todo.md` is listed twice in the status. The first change is staged and ready to be committed. The second change (adding the heading) is unstaged. If we were to commit right now, then the comment would not be saved in the repository. Let us do so.

```
git commit -m "Add third todo item"
git status
```

Output:

```
+ git commit -m 'Add third todo item'
[master 6d5604d] Add third todo item
 1 files changed, 1 insertions(+), 0 deletions(-)
+ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   todo.md
#
no changes added to commit (use "git add" and/or "git commit -a")
```

The status command indicates that `todo.md` has unrecorded changes, but is no longer in the staging area.

Add the second change to the staging area, then check the status.

```
git add todo.md
git status
```

Output:

```
+ git add todo.md
+ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   todo.md
#
```

The second change has been staged and is ready to commit.

Execute:

```
git commit -m "Add a heading"
```

Output:

```
+ git commit -m 'Add a heading'
[master 676e06d] Add a heading
 1 files changed, 2 insertions(+), 0 deletions(-)
```

8 History

The `git log` command shows a listing of what changes have been made.

Execute:

```
git log
```

Output:

```
+ git log
commit 676e06d694f7e0dc2b9f6662ace77d9aa25134d8
Author: Dr. Schwesinger <schwesin@kutztown.edu>
Date:   Sun Jan 26 19:43:40 2020 -0500

    Add a heading

commit 6d5604d5a7a9040e28344d9b5395f0626217fb4b
Author: Dr. Schwesinger <schwesin@kutztown.edu>
Date:   Sun Jan 26 19:43:40 2020 -0500

    Add third todo item

commit 2ef381dd3aed614dc586209ab5360777cb09aa80
Author: Dr. Schwesinger <schwesin@kutztown.edu>
Date:   Sun Jan 26 19:43:39 2020 -0500

    Add second todo item

commit fe1c3e49e5b7edfd4a3a3ed7fcd9f90594f33f3e
Author: Dr. Schwesinger <schwesin@kutztown.edu>
Date:   Sun Jan 26 19:43:38 2020 -0500

    Initial Commit
```

Here is a set of all four commits that we have made to the repository so far.

The `git log` command has many options for formatting the output. For example,

```
+ git log --pretty=oneline
676e06d694f7e0dc2b9f6662ace77d9aa25134d8 Add a heading
6d5604d5a7a9040e28344d9b5395f0626217fb4b Add third todo item
2ef381dd3aed614dc586209ab5360777cb09aa80 Add second todo item
fe1c3e49e5b7edfd4a3a3ed7fcd9f90594f33f3e Initial Commit
```

9 Getting old versions

The `git checkout` command will copy any snapshot from the repository to the working directory.

The output from the `git log` command contains a hash code for each commit. The hash code is the 40 character long string of alphanumeric characters. Examine the log and find the hash code of the initial commit. Your commit hash codes may not match the example above.

Execute:

```
git checkout <hash>
cat todo.md
```

The first 7 characters of the hash code should be sufficient for this command.

Output:

```
+ git checkout fe1c3e4
Note: checking out 'fe1c3e4'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at fe1c3e4... Initial Commit
+ cat todo.md
* Do homework
```

Note that the contents of the `todo.md` file are the original contents.

To return to the latest version execute the command:

```
git checkout master
cat todo.md
```

Output:

```
+ git checkout master
Previous HEAD position was fe1c3e4... Initial Commit
Switched to branch 'master'
+ cat todo.md
# TODO list

* Do homework
* Buy groceries
* Study for exam
```

Here “master” is the name of the default branch.

10 Tagging versions

Let us call the current version of the project version 1 (v1).

Execute:

```
git tag v1
```

The `v1` tag points to the current commit. We can create one (or more) tags to any commit.

We can use the name `v1` to access the commit instead of using the hash code.

Execute:

```
git checkout v1
```

Output:

```
+ git tag v1
+ git checkout v1
Note: checking out 'v1'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b new_branch_name

HEAD is now at 676e06d... Add a heading
```

We can view the available tags by using the `git tag` command without any arguments.

Execute:

```
git tag
```

Output:

```
+ git tag
v1
```

11 Undoing Local Changes (before staging)

Make sure that you are on the latest commit before proceeding.

Execute:

```
git checkout master
```

Sometimes you have modified a file in your local working directory and you want to revert the file to the latest commited version. The `git checkout` command will perform that change.

Change the `todo.md` file as follows:

File: `todo.md`

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
* Item we added by mistake
```

Check the status of the working directory.

Execute:

```
git status
```

Output:

```
+ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   todo.md
#
no changes added to commit (use "git add" and/or "git commit -a")
```

The `todo.md` file has been modified, but it has not been staged yet.

Use `git checkout` command to revert the change to the repository's version of the `todo.md` file.

Execute:

```
git checkout todo.md
git status
cat todo.md
```

Output:

```
+ git checkout todo.md
+ git status
```

```
# On branch master
nothing to commit (working directory clean)
+ cat todo.md
# TODO list

* Do homework
* Buy groceries
* Study for exam
```

The `git status` command shows us that there are no outstanding changes in the working directory. Also, note that the mistakenly added todo item is no longer in the `todo.md` file.

12 Undoing Staged Changes (before committing)

Change the `todo.md` again to have an unwanted item:

File: `todo.md`

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
* Item we added by mistake
```

Stage the change with the `git add` command.

Execute:

```
git add todo.md
```

Check the status.

Execute:

```
git status
```

Output:

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

The output of `git status` shows that the change has been staged. Note that the output also indicates how to unstage the change.

Execute:

```
git reset HEAD todo.md
```

Output:

```
+ git reset HEAD todo.md
Unstaged changes after reset:
M       todo.md
```

The `git reset` command resets the staging area to be whatever is in `HEAD`. The `git reset` command does not change the working directory by default. So, the working directory still has the unwanted change.

Use the `git checkout` command from the previous step to remove the unwanted change from the working directory.

Execute:

```
git checkout todo.md
git status
```

Output:

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

The working directory is now clean.

13 Moving Files

Here we are going to change the structure of our repository. Let us move the `todo.md` into a directory named `documents`.

Execute:

```
mkdir documents
git mv todo.md documents
git status
```

Output:

```
+ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       renamed:    todo.md -> documents/todo.md
#
```

By using git to do the move, we inform git of two things

1. The `todo.md` file has been deleted
2. The file `documents/todo.md` has been created

This information is immediately staged and ready to be committed. The git status command reports that the file has been moved (renamed).

There is an alternative method of accomplishing the same task:

```
mkdir documents
mv todo.md documents
git add documents/todo.md
git rm todo.md
```

Let us commit this move.

Execute:

```
git commit -m "Move todo.md to documents"
```

Output:

```
+ git commit -m 'Move todo.md to documents'
[master 6f6f6f5] Move todo.md to documents
 1 files changed, 0 insertions(+), 0 deletions(-)
 rename todo.md => documents/todo.md (100%)
```

14 Creating a Branch

Let us make some experimental changes to our existing project. We do not know if these changes will be worthwhile, so we should move these changes into a new branch to isolate them from the master branch changes.

Let us name our new branch “`experimental`”.

Execute:

```
git checkout -b experimental
git status
```

Output:

```
+ git checkout -b experimental
Switched to a new branch 'experimental'
+ git status
# On branch experimental
nothing to commit (working directory clean)
```

Note that the `git status` command reports that we are on the `experimental` branch.

Add a new file named `documents/notes.md` with the content:

File: `notes.md`

```
# Notes

* a = mx + b
```

Execute:

```
git add documents/notes.md
git commit -m "Add notes.md"
```

Update the `documents/todo.md` file with the content:

File: `todo.md`

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
* Add more notes to notes.md
```

Execute:

```
git add documents/todo.md
git commit -m "Add fourth todo item"
```

We now have a new branch named experimental with two new commits.

15 Navigating Branches

The project now has two branches.

Execute:

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

Output:

```
+ git log --pretty=oneline
06b4aebef43a3f8626d743c4d6a59730fd9cd0f6 Add fourth todo item
b59ff71a409789788b252b6c9687cc8448ec2e20 Add notes.md
6f6f6f5642d5c34ce6813288ec644a1c2b4c2575 Move todo.md to documents
676e06d694f7e0dc2b9f6662ace77d9aa25134d8 Add a heading
6d5604d5a7a9040e28344d9b5395f0626217fb4b Add third todo item
2ef381dd3aed614dc586209ab5360777cb09aa80 Add second todo item
fe1c3e49e5b7edfd4a3a3ed7fcd9f90594f33f3e Initial Commit
```

The `git checkout` command is used to switch between branches.

Execute:

```
git checkout master
cat documents/todo.md
```

Output:

```
+ git checkout master
Switched to branch 'master'
+ cat documents/todo.md
# TODO list

* Do homework
* Buy groceries
* Study for exam
```

To get back to the experimental branch, use another checkout command.

Execute:

```
git checkout experimental
cat documents/todo.md
```

Output:

```
Switched to branch 'experimental'
+ cat documents/todo.md
# TODO list

* Do homework
* Buy groceries
* Study for exam
* Add more notes to notes.md
```

16 Changes to master branch

Suppose that we need to make a change to the master branch not related to the experimental branch.

Switch to the master branch.

Execute:

```
git checkout master
```

Create a file named `README` with the content:

File: `README`

```
This is a README file for the tutorial project.
```

Commit it to the master branch.

Execute:

```
git add README
git commit -m "Add README"
```

We can get a text based representation of the branches by supplying the `--graph` and `--all` flags to the `git log` command.

Execute:

```
git log --pretty=oneline --graph --all
```

Output:

```
+ git log --pretty=oneline --graph --all
* 8da1e7014ea89be2fd00eb8755b16f77c71fccc7 Add README
| * 06b4aebef43a3f8626d743c4d6a59730fd9cd0f6 Add fourth todo item
| * b59ff71a409789788b252b6c9687cc8448ec2e20 Add notes.md
|/
* 6f6f6f5642d5c34ce6813288ec644a1c2b4c2575 Move todo.md to documents
* 676e06d694f7e0dc2b9f6662ace77d9aa25134d8 Add a heading
* 6d5604d5a7a9040e28344d9b5395f0626217fb4b Add third todo item
* 2ef381dd3aed614dc586209ab5360777cb09aa80 Add second todo item
* fe1c3e49e5b7edfd4a3a3ed7fcd9f90594f33f3e Initial Commit
```

17 Merging

Merging brings changes from two branches into one branch. Go back to the experimental branch and merge it with master.

Execute:

```
git checkout experimental
git merge master
git log --pretty=oneline --graph --all
```

Output:

```
+ git checkout experimental
Switched to branch 'experimental'
+ git merge master
Merge made by recursive.
 README |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README
+ git log --pretty=oneline --graph --all
*   29a4f4388c3e4f7052ada41eca18624c7d8523b5 Merge branch 'master' into experimental
|\
| * 8da1e7014ea89be2fd00eb8755b16f77c71fccc7 Add README
* | 06b4aebef43a3f8626d743c4d6a59730fd9cd0f6 Add fourth todo item
* | b59ff71a409789788b252b6c9687cc8448ec2e20 Add notes.md
|/
* 6f6f6f5642d5c34ce6813288ec644a1c2b4c2575 Move todo.md to documents
* 676e06d694f7e0dc2b9f6662ace77d9aa25134d8 Add a heading
* 6d5604d5a7a9040e28344d9b5395f0626217fb4b Add third todo item
* 2ef381dd3aed614dc586209ab5360777cb09aa80 Add second todo item
* fe1c3e49e5b7edfd4a3a3ed7fcd9f90594f33f3e Initial Commit
```

The experimental branch has incorporated the changes made to the master branch.

18 Creating a conflict

Return to the master branch and change the `documents/todo.md` file.

Execute:

```
git checkout master
```

Change `documents/todo.md`.

File: `todo.md`

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
* Do laundry
```

Execute:

```
git add documents/todo.md
git commit -m "Add fourth todo item"
```

View the branches.

Execute:

```
git log --pretty=oneline --graph --all
```

Output:

```
+ git log --pretty=oneline --graph --all
* e91eb20362457e411abc1126d2e5e6d388ae3662 Add fourth todo item
| *   29a4f4388c3e4f7052ada41eca18624c7d8523b5 Merge branch 'master' into experiment
| |\
| |/
|/|
* | 8da1e7014ea89be2fd00eb8755b16f77c71fccc7 Add README
| * 06b4aebef43a3f8626d743c4d6a59730fd9cd0f6 Add fourth todo item
| * b59ff71a409789788b252b6c9687cc8448ec2e20 Add notes.md
|/
* 6f6f6f5642d5c34ce6813288ec644a1c2b4c2575 Move todo.md to documents
* 676e06d694f7e0dc2b9f6662ace77d9aa25134d8 Add a heading
* 6d5604d5a7a9040e28344d9b5395f0626217fb4b Add third todo item
* 2ef381dd3aed614dc586209ab5360777cb09aa80 Add second todo item
* fe1c3e49e5b7edfd4a3a3ed7fcd9f90594f33f3e Initial Commit
```

19 Resolving conflicts

Let us go back to the experimental branch and merge it with the new master branch.

Execute:

```
git checkout experimental
git merge master
```

Output:

```
+ git checkout experimental
Switched to branch 'experimental'
+ git merge master
Auto-merging documents/todo.md
CONFLICT (content): Merge conflict in documents/todo.md
Automatic merge failed; fix conflicts and then commit the result.
```

If you open the `documents/todo.md` you will see:

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
<<<<<<< HEAD
* Add more notes to notes.md
=======
* Do laundry
>>>>>>> master
```

The conflicting text is demarcated with `<<<<<<< HEAD` and `>>>>>>> master`. The first section is the version in the current branch. The second section is the version of the master branch.

In this case, the conflict needs to be resolved manually. Edit the `documents/todo.md` file to be:

```
# TODO list

* Do homework
* Buy groceries
* Study for exam
* Add more notes to notes.md
* Do laundry
```

Then we can make a commit of the conflict resolution.

Execute:

```
git add documents/todo.md
git commit -m "Merge master, fix conflict"
```

Change back to the master branch. Execute:

```
git checkout master
```

20 Cloning Repositories

Here we will make a copy of a repository.

Go to the working directory and clone your tutorial repository.

Execute:

```
cd ..
```

Your current working directory should contain the `tutorial` repository. Now let us create a clone of the repository.

Execute:

```
git clone tutorial cloned_tutorial
```

Output:

```
+ cd ..
+ git clone tutorial cloned_tutorial
Initialized empty Git repository in .../work/cloned_tutorial/.git/
```

21 Examine the cloned repository

Let us have a look at our cloned repository.

Execute:

```
cd cloned_tutorial
ls
```

Output:

```
+ cd cloned_tutorial
+ ls
documents
README
```

If you view the history with `git log`, you will see a list of all the commits in the new repository and it should match the commit history of the original repository. The only difference should be the names of the branches. That is, some of the branch names are prefixed with `origin/`.

22 What is origin?

Execute:

```
git remote
```

Output:

```
+ git remote
origin
```

We see that the cloned repository knows the default name of the remote repository. To get more information about origin:

Execute:

```
git remote show origin
```

Output:

```
+ git remote show origin
* remote origin
  Fetch URL: .../work/tutorial
  Push  URL: .../work/tutorial
  HEAD branch: master
  Remote branches:
    experimental 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)
```

We can see that “origin” of the remote repository is the original `tutorial` repository. Remote repositories are typically stored on a separate machine or centralized server. However, they can also point to a repository on the same machine. There is nothing special aout the name “origin”, but there is a convention to use it for the primary centralized repository.

23 Remote branches

Let us take a look at the branches in our cloned repository.

Execute:

```
git branch
```

Output:

```
+ git branch
* master
```

The only branch listed is master; the experimental branch is not listed. By default, `git branch` only lists local branches. To see all branches the `-a` flag can be used.

Execute:

```
git branch -a
```

Output:

```
+ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/experimental
  remotes/origin/master
```

Git lists all the branches fro the original repository, but the remote branches are not treated as local branches. If we need our own experimental branch, then we need to create it ourselves.

24 Changing the original repository

First change directories to the original repository.

Execute:

```
cd ../tutorial
```

Make the following changes to the README file:

```
This is a README file for the tutorial project.
(changed in original)
```

Commit the change.

Execute:

```
git add README
git commit -m "Change README in original repository"
```

Output:

```
+ git add README
+ git commit -m 'Change README in original repository'
[master a0e9346] Change README in original repository
 1 files changed, 1 insertions(+), 0 deletions(-)
```

25 Fetching changes

Execute:

```
cd ../cloned_tutorial
git fetch
```

Output:

```
+ cd ../cloned_tutorial
+ git fetch
From /home/KUTZTOWN/schwesin/private/work/tutorial
   e91eb20..a0e9346  master      -> origin/master
```

At the moment, the repository contains all the commits from the original repository, but they are not yet integrated into the local branches of the cloned repository. The `git log` command shows that the commit named "Change README in original repository" is in the history. But, note that this commit is not pointed to by master. You can verify this by viewing the content of the `README` file.

26 Merging fetched changes

Execute:

```
git merge origin/master
```

Output:

```
+ git merge origin/master
Updating e91eb20..a0e9346
Fast-forward
 README |    1 +
 1 files changed, 1 insertions(+), 0 deletions(-)
```

The changes are now integrated into the current branch.

Execute:

```
cat README
```

Output:

```
+ cat README
This is a README file for the tutorial project
(changed in original)
```

27 Pull

The `git pull` command is identical to:

```
git fetch
git merge
```

So, `git pull` is a convenient shortcut.

28 Adding a Tracking Branch

Execute:

```
git branch -a
```

Output:

```
+ git branch -a
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/experimental
  remotes/origin/master
```

The branches starting with `remotes/origin` are branches from the original repository. Notice that the cloned repository does not have a branch named experimental, but it knows that the original repository does have a branch named experimental.

Add a local branch that tracks a remote branch.

Execute:

```
git branch --track experimental origin/experimental
git branch -a
```

Output:

```
+ git branch --track experimental origin/experimental
Branch experimental set up to track remote branch experimental from origin.
+ git branch -a
  experimental
* master
  remotes/origin/HEAD -> origin/master
  remotes/origin/experimental
  remotes/origin/master
```

We can see that experimental branch is now listed.

29 Bare repositories

A bare repository is usually used for sharing.

Create a bare repository.

Execute:

```
cd ..
git clone --bare tutorial tutorial.git
ls tutorial.git
```

Output:

```
+ cd ..
+ git clone --bare tutorial tutorial.git
Initialized empty Git repository in .../work/tutorial.git/
+ ls tutorial.git
branches
config
description
HEAD
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 `tutorial.git` repository. Essentially it is nothing but the `.git` directory of a non-bare repository.

30 Adding a remote repository

Add the `tutorial.git` repository to our original repository.

```
cd tutorial
git remote add shared ../tutorial.git
```

31 Submitting changes

Bare repositories are usually shared on a network server. Since we cannot typically access the file system of the network server, we need a different way to submit changes to the shared repository.

Change the `README` file:

```
This is a README file for the tutorial project.
(changed in original and pushed to shared)
```

Execute:

```
git add README
git commit -m 'Add change to README'
```

Push the change to the shared repository.

Execute:

```
git push shared master
```

Output:

```
+ git push shared master
To ../tutorial.git
   a0e9346..fd3e8c1  master -> master
```

The name of the repository receiving the changes is named “shared”. This is what we named it in the previous step.

32 Pulling shared changes

Change to the `cloned_tutorial` repository and pull the changes that were just pushed to the shared repository.

Execute:

```
cd ../cloned_tutorial
git remote add shared ../tutorial.git
git branch --track shared master
git pull shared master
```

Output:

```
+ cd ../cloned_tutorial
+ git remote add shared ../tutorial.git
+ git branch --track shared master
Branch shared set up to track local branch master.
+ git pull shared master
From .../work/tutorial
   a0e9346..fd3e8c1  master     -> origin/master
Updating a0e9346..fd3e8c1
Fast-forward
 README |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```