How to clone all remote branches in Git?

Asked 11 years, 10 months ago Active 20 days ago Viewed 1.3m times

4161

I have a `master` and a `development` branch, both pushed to GitHub. I've `clone`d, `pull`ed, and `fetch`ed, but I remain unable to get anything other than the `master` branch back.

I'm sure I'm missing something obvious, but I have read the manual and I'm getting no joy at all.

`git` `github` `git-branch` `git-clone` `remote-branch`

1804

share improve this question follow

edited Dec 15 '17 at 14:52

Md. Abu Nafee Ibna Zahid

435 ●1 ●5 ●15

asked Sep 15 '08 at 22:42

Peter Coulton

48.4k ●11 ●50 ●69

135 The accepted answer here (`git branch -a`) shows you the branches in the remote, but if you attempt to check any of those out you will be in a 'detached HEAD' state. The next answer down (second most upvotes) answers a different question (to wit: how to pull all branches, and, again, this only works for those you're tracking locally). Several of the comments point out that you could parse the `git branch -a` results with a shell script that would locally track all the remote branches. Summary: There's no git native way to do what you want and it might not be all that great an idea anyway. – Day Davis Waterbury Jun 18 '12 at 22:43

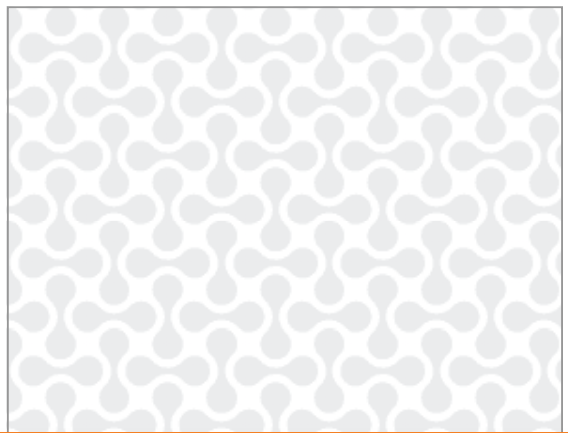
5 Maybe just copy the entire folder the old fashioned way? `scp some_user@example.com:/home/some_user/project_folder ~` Not sure if that solution works for github though.. – snapfractalpop Sep 26 '12 at 22:51

19 Rather than saying "I've cloned, pulled, and fetched," much better to show us the *exact commands* that you executed. – Bob Gilmore Nov 22 '13 at 18:17

58 It always boggles me why "clone" isn't in the sense of an exact copy. If it's an exact clone, shouldn't all the branches be part of the local repository? I mean isn't that one of the point of being distributed? So when something repository is gone you still have a complete copy of everything. Or is it the so called "remote" really are part of the local repository already? – huggie Jul 11 '16 at 6:31

21 Seeing all the upvotes, answers, comments on answers and the mind-boggling number of views, I think it is time git added a command for doing this. And right you are @huggie, my thoughts exactly. – Знаđошfаχ Aug 29 '16 at 4:29

show 8 more comments



38 Answers

Active Oldest Votes

1 2 Next

4580

1

First, clone a remote Git repository and cd into it:

```
$ git clone git://example.com/myproject
$ cd myproject
```

Next, look at the local branches in your repository:

```
$ git branch
* master
```

But there are other branches hiding in your repository! You can see these using the `-a` flag:

```
$ git branch -a
* master
  remotes/origin/HEAD
  remotes/origin/master
  remotes/origin/v1.0-stable
  remotes/origin/experimental
```

If you just want to take a quick peek at an upstream branch, you can check it out directly:

```
$ git checkout origin/experimental
```

But if you want to work on that branch, you'll need to create a local tracking branch which is done automatically by:

```
$ git checkout experimental
```

and you will see

```
Branch experimental set up to track remote branch experimental from origin.
Switched to a new branch 'experimental'
```

That last line throws some people: "New branch" - huh? What it really means is that the branch is taken from the index and created locally for you. The *previous* line is actually more informative as it tells you that the branch is being set up to track the remote branch, which usually means the origin/branch_name branch

Now, if you look at your local branches, this is what you'll see:

```
$ git branch
* experimental
  master
```

You can actually track more than one remote repository using `git remote` .

```
$ git remote add win32 git://example.com/users/joe/myproject-win32-port
$ git branch -a
* master
  remotes/origin/HEAD
  remotes/origin/master
  remotes/origin/v1.0-stable
  remotes/origin/experimental
  remotes/win32/master
  remotes/win32/new-widgets
```

At this point, things are getting pretty crazy, so run `gitk` to see what's going on:

```
$ gitk --all &
```

share improve this answer follow

edited Apr 25 '17 at 8:57

sebastian-c
13.6k ● 3 ● 40 ● 86

answered Sep 16 '08 at 13:28

emk
53.9k ● 6 ● 37 ● 48

120 How can someone create automatically all the remote branches, e.g. experimental for origin/experimental? – Cristian Ciupitu Jun 4 '09 at 16:33

54 Cristian: I used to always create a branch 'foo' for every branch 'origin/foo', but this led to two problems: (1) I wound up with lots of really stale tracking branches that were many commits behind the corresponding remote branch, and (2) in older versions of git, running 'git push' would attempt to push all my local branches to a remote, even when those branches were stale. So now I only keep local branches for things that I'm actively developing, and access the origin/* branches directly if I need information about them. (That said, you could use a shell script to parse 'git branch -a'.) – emk Jul 20 '09 at 21:44

49 "git fetch <origin-name> <branch-name>" brings the branch down locally for you. – user67627 Feb 7 '10 at 17:10

140 Good answer, but kinda misses the question. I was looking for a one-liner to checkout all the remote branches. – cmcginty Oct 19 '10 at 21:01

32 The question was about cloning all remote branches, not checking them out. And, as I noted above, you really don't want to make any more local tracking branches than necessary, because when they get really stale, they can cause headaches. – emk Oct 28 '10 at 12:43

show 21 more comments

828

If you have many remote branches that you want to fetch at once, do:

```
$ git pull --all
```

Now you can checkout any branch as you need to, without hitting the remote repository.

1

share improve this answer follow

edited Feb 6 '15 at 20:04

answered Jan 13 '11 at 16:42

Gabe Kopley
14.3k ● 5 ● 39 ● 56

12 If I do git clone, I have the master branch locally and 10 branches "remote". So THIS answer by Gabe was very helpful and answers the question. – basZero Jan 27 '12 at 19:07

41 this only fetch remote branches that have been locally added not *any* remote branch – jujule Feb 10 '12 at 11:45

33 The first command is redundant. Simply `git pull --all` will do the same – it just won't fetch twice. And infosec812 is

6 After I did `git remote update`, then tried `git branch`, I only see local branches. But if I do `git branch -a` I can now see the remote branches and I can do a `git pull <branchname>` to get the branch I want. -- I landed on this question from a Google search, and this answer solves my problem. – WNRosenberg Jan 31 '13 at 16:47

40 This is not helpful at all, doesn't pull any remote branches other than that is existing. – Avinash R Apr 11 '13 at 15:02

show 8 more comments

452

This Bash script helped me out:

```
#!/bin/bash
for branch in $(git branch --all | grep '^\s*remotes' | egrep --invert-match '(:?HEAD|master)$'); do
    git branch --track "${branch##*/}" "$branch"
done
```

It will create tracking branches for all remote branches, except master (which you probably got from the original clone command). I think you might still need to do a

```
git fetch --all
git pull --all
```

to be sure.

> One liner: `git branch -a | grep -v HEAD | perl -ne 'chomp($_); s|^\*?\s*||; if (m|(.+)/(.+)| && not $d{$2}) {print qq(git branch --track $2 $1/$2\n)} else {$d{$_}=1}' | csh -xfs`
> As usual: test in your setup before copying rm -rf universe as we know it
>
> *Credits for one-liner go to user cfi*

share improve this answer follow

edited Nov 12 '19 at 14:56 vinzee 6,498 ● 9 ● 30 ● 47

answered Jan 21 '11 at 2:18 bigfish 4,583 ● 1 ● 11 ● 3

19 This is really close to being a perfect solution.. The only thing that would make it better is if this functionality were built-in as an option in git. – Deven Phillips Oct 27 '11 at 17:15

51 "One liner": `git branch -a | grep -v HEAD | perl -ne 'chomp($_); s|^\*?\s*||; if (m|(.+)/(.+)| && not $d{$2}) {print qq(git branch --track $2 $1/$2\n)} else {$d{$_}=1}' | csh -xfs` As usual: test in your setup before copying `rm -rf` universe as we know it – cfi Sep 18 '12 at 12:38

4 This command creates the feature branches from remote as normal branches (not feature branches) - how to fix this? – Alex2php Mar 20 '14 at 14:31

4 if you run into issues with "/" in branch names there is a solution below using a git alias. see answer by "nobody" on "answered May 15 '13 at 11:02" – wemu Jan 12 '15 at 11:54

8 I'm trimming just `remotes/origin/` to preserve namespaces: `for BRANCH in $(git branch -a | grep remotes | grep -v HEAD | grep -v master); do git branch --track "${BRANCH#remotes/origin/}" "${BRANCH}"; done` – kgadek Jul 2 '15 at 8:33

show 7 more comments

353

Using the `--mirror` option seems to copy the `remote` tracking branches properly. However, it sets up the repository as a bare repository, so you have to turn it back into a normal repository afterwards.

```
git clone --mirror path/to/original path/to/dest/.git
cd path/to/dest
git config --bool core.bare false
git checkout anybranch
```

Reference: Git FAQ: How do I clone a repository with all remotely tracked branches?

share improve this answer follow

edited Oct 19 '15 at 9:21
olibre
36.9k ● 23 ● 135 ● 177

answered Aug 27 '11 at 17:49
Dave
3,555 ● 1 ● 10 ● 2

8 You know this actually seems to be a pretty good answer even though it has no votes. Are there any pitfalls to doing it that way? I had to explicitly checkout a branch after running those commands. – loop Jan 12 '12 at 4:25

27 This combined with git push --mirror are exactly what I needed to create an exact duplicate of a remote git repo when moving from github.com to a github enterprise install. Thanks! – Jacob Fike Sep 11 '12 at 22:59

4 @Dave: Add a final `git checkout` as last command to finally checkout the head of the current branch on the cloned repo. This is a great answer, by far the best. Be brave, eventually we'll get you to the top :-) – cfi Sep 18 '12 at 7:37

3 @Dave: Hm. I'm having second thoughts: --mirror does more than just setting up all branches as being tracked. It copies all refs from the origin and subsequent `git remote update` will do that again. Behaviour of pulls change. I'm back to believing the full copy requires a one-line script. – cfi Sep 18 '12 at 11:53

6 `git clone --mirror` is very good for backing up your git repositories ^_^ – TrinitronX May 20 '13 at 21:19

show 11 more comments

221

You can easily switch to a branch without using the fancy "git checkout -b somebranch origin/somebranch" syntax. You can just do:

```
git checkout somebranch
```

Git will automatically do the right thing:

```
$ git checkout somebranch
Branch somebranch set up to track remote branch somebranch from origin.
Switched to a new branch 'somebranch'
```

Git will check whether a branch with the same name exists in exactly one remote, and if it does, it tracks it the same way as if you had explicitly specified that it's a remote branch. From the git-checkout man page of Git 1.8.2.1:

> If <branch> is not found but there does exist a tracking branch in exactly one remote (call it <remote>) with a matching name, treat as equivalent to
>
> ```
> $ git checkout -b <branch> --track <remote>/<branch>
> ```


44.6k ● 8 ● 52 ● 84

1 So, if the name of the branch you `checkout` is the identical to the name of the remote branch, everything after the "/", then git will create a branch of the same name, everything after the "/", "tracking" that remote? And by tracking, we mean: `git push`, `git pull`, etc. will be done on that remote? If this is correct, then expand on your answer with more information, because I aggree with @Daniel, this answer deserves more rep. – Gerard Roche Jun 14 '12 at 22:08

5 @BullfrogBlues, the answer to all your questions appears to be yes (I'm using git v1.7.7.4). I agree this behavior should be better known. (It's not in the manual for this version of git.) I actually don't like this behavior, I'd rather get an error and have to say `git checkout --track origin/somebranch` explicitly. – dubiousjim Jul 3 '12 at 13:05

3 @dubiousjim: Actually, this is in the manual. git-checkout(1) says: "If <branch> is not found but there does exist a tracking branch in exactly one remote (call it <remote>) with a matching name, treat as equivalent to 'git checkout -b <branch> --track <remote>/<branch>' " (Git V.1.8.1.1). – sleske Feb 27 '13 at 9:01

1 What we need is $ git pull * <remote>/* - where "*" is a wildcard, so it pulls all branches, including those not yet on the local system. How to do this? Are we really supposed to checkout/pull every branch just to get the code pulled to our local system? – JosephK Feb 19 '17 at 5:27

add a comment

99

Regarding,

> $ git checkout -b experimental origin/experimental

using

```
$ git checkout -t origin/experimental
```

or the more verbose but easier to remember

```
$ git checkout --track origin/experimental
```

might be better, in terms of tracking a remote repository.

share improve this answer follow

edited Jul 9 '13 at 5:05


xverges
3,776 ● 1 ● 32 ● 55

answered Jul 27 '09 at 6:10


murphytalk
1,233 ● 9 ● 11

So you mean the second form is only easier to remember and there's no other difference? – aderchox Jul 18 '19 at 15:09

add a comment

79

The fetch that you are doing should get all the remote branches, but it won't create local branches for them. If you use gitk, you should see the remote branches described as "remotes/origin/dev" or something similar.

To create a local branch based on a remote branch, do something like:

```
git checkout -b dev refs/remotes/origin/dev
```

```
Branch dev set up to track remote branch refs/remotes/origin/dev.
Switched to a new branch "dev"
```

Now, when you are on the dev branch, "git pull" will update your local dev to the same point as the remote dev branch. Note that it will fetch all branches, but only pull the one you are on to the top of the tree.

share improve this answer follow

answered Sep 15 '08 at 22:52

Luuk Paulussen
853 ●1 ●6 ●7

14 You don't need refs/remotes here. git checkout -b dev origin/dev will work fine. – emk Sep 17 '08 at 13:10

3 This will always work: `git checkout -b newlocaldev --track origin/dev`. If you want the local branch has the same name as the remote one, and the remote one doesn't have a tricky name, you can omit the `-b newlocaldev`. With the default `branch.autosetupmerge` config setting, and assuming you don't have a local branch named `dev`, these two commands may do the same thing: `git checkout -b dev origin/dev` and just plain `git checkout dev`. Finally, `git checkout origin/dev` doesn't create a new branch, but just puts you in detached HEAD state. – dubiousjim Jul 3 '12 at 13:27

What happens when the remote no longer exists but Git is too stupid to acknowledge its been deleted? This assumes you updated and `git branch -a` continues to lists it as a remote branch. – jww Sep 19 '16 at 22:14

1 And we do this for dozens of branches? – JosephK Feb 19 '17 at 5:28

add a comment

59

When you do "git clone git://location", all branches and tags are fetched.

In order to work on top of a specific remote branch, assuming it's the origin remote:

```
git checkout -b branch origin/branchname
```

share improve this answer follow

answered Sep 15 '08 at 22:47

elmarco
25.7k ●21 ●56 ●68

2 I appreciate your note "all branches and tags are fetched". I was going to comment on your answer being wrong, but then I checked it and found, you are perfectly right. So in a way, you have provided the shortest answer - if you cloned, you already have it. Nice. One could try to add: try `$ git branch -a` to learn, what remote branches are already available. – Jan Vlcinsky Nov 17 '13 at 22:39

Have a look at the answer I posted as well. This might be helpful if you know you'll want to be working locally on many of the remote branches and not have to check them out one by one. – lacostenycoder Dec 8 '18 at 15:39

Can you explain to me what's the difference between `git checkout -b master origin/master` and `git checkout --track origin/master` please ? – aderchox Jul 18 '19 at 15:08

1 @aderchox Nowadays, I think none. – elmarco Aug 2 '19 at 10:05

add a comment

59

Use aliases. Though there aren't any native Git one-liners, you can define your own as

```
git config --global alias.clone-branches '! git branch -a | sed -n "/\/HEAD /d; /\/master$/d; /remotes/p;" | xargs -L1 git c
```

```
git clone-branches
```

share improve this answer follow

edited Jun 16 '13 at 13:48

Peter Mortensen
26.3k ● 21 ● 91 ● 121

answered May 15 '13 at 11:02

nobody
599 ● 4 ● 3

1 Thanks. This actually clones all remote branches unlike several of the other answers – FearlessHyena May 14 at 10:22

add a comment

51

Better late than never, but here is the best way to do this:

```
mkdir repo
cd repo
git clone --bare path/to/repo.git .git
git config --unset core.bare
git reset --hard
```

1

At this point you have a complete copy of the remote repo with all of it's branches (verify with `git branch`). You can use `--mirror` instead of `--bare` if your remote repo has remotes of its own.

share improve this answer follow

edited May 11 '17 at 21:08

answered Nov 26 '12 at 23:42

Jacob Fike
731 ● 5 ● 7

Something went wrong during the edits here. Now this answer doesn't make sense. The " `--bare` " mentioned in the last sentence doesn't exist in the given command list. – Cerran Mar 5 '14 at 13:11

taking from Dave's answer below. Using 'git config --bool core.bare false' instead of 'git config unset core.bare' seems to do the job. – Confused Vorlon Oct 28 '14 at 12:25

I have the `error: key does not contain a section: unset`. The Dave's answer works better. – olibre Oct 19 '15 at 9:24

It's `git config --unset core.bare` actually... To me, this seems the cleanest solution of all presented in the answers here. A pity it has so few upvotes... – Dirk Hillbrecht May 10 '17 at 8:05

1 Thanks @ChrisSim I agree for `git config --bool core.bare false`. This is why I recommend instead the Dave's answer. What do you think about the Dave's answer? Cheers – olibre Jun 29 '17 at 8:24

show 2 more comments

51

Why you only see "master"

`git clone` downloads all remote branches but still considers them "remote", even though the files are located in your new repository. There's one exception to this, which is that the cloning process creates a local branch called "master" from the remote branch called "master". By default, `git branch` only shows local branches, which is why you only see "master".

1

`git branch -a` shows all branches, *including remote branches*.

How to get local branches

working directory), you can do that like this:

```
git branch branchone origin/branchone
git branch branchtwo origin/branchtwo
git branch branchthree origin/branchthree
```

In this example, `branchone` is the name of a local branch you're creating based on `origin/branchone`; if you instead want to create local branches with different names, you can do this:

```
git branch localbranchname origin/branchone
```

Once you've created a local branch, you can see it with `git branch` (remember, you don't need `-a` to see local branches).

share improve this answer follow

edited Apr 2 at 18:14

Niroshan Ratnayake
118 ● 1 ● 9

answered Mar 5 '14 at 13:47

Cerran
1,657 ● 2 ● 16 ● 28

If `origin/branchone` exists, you can also just use `git checkout branchone` to create a local branch with the same name and set it to track remote. – Evan May 15 at 20:38

add a comment

47

This isn't too much complicated, very simple and straight forward steps are as follows;

`git fetch origin` This will bring all the remote branches to your local.

`git branch -a` This will show you all the remote branches.

`git checkout --track origin/<branch you want to checkout>`

Verify whether you are in the desired branch by the following command;

```
git branch
```

The output will like this;

```
*your current branch
some branch2
some branch3
```

Notice the * sign that denotes the current branch.

share improve this answer follow

edited Jun 9 '16 at 5:16

answered Dec 26 '13 at 10:19

Sam
1,343 ● 1 ● 15 ● 26

1 Thanks suraj. The reason because its not been voted much. And the ans is not accepted by the questioner. – Sam Jun 8 '15 at 11:03

The "git fetch origin" did not bring any of the remote branches to my local - or are they hidden somewhere? Reading all the answers above gave me a headache. We are looking for "git fetch all branches to local". There must be a way

branch_name -> origin/branch_name", but when you run "git branch" it will show you only your local branches instead, so to see all the branches you can do "git branch -a" and then to switch to the remote branch you need to run " git checkout --track origin/<branch you want to checkout>". Hope this helps. :-) – Sam Jun 9 '16 at 4:24

4 Suraj, because the question was, how to "clone all remote branches" - not how to manually update one at a time. It appears there is no answer to the actual question - just ways to do a whole lot of typing if you have a lot of branches. – JosephK Feb 19 '17 at 7:38

add a comment

39

Just do this:

```
$ git clone git://example.com/myproject
$ cd myproject
$ git checkout branchxyz
Branch branchxyz set up to track remote branch branchxyz from origin.
Switched to a new branch 'branchxyz'
$ git pull
Already up-to-date.
$ git branch
* branchxyz
  master
$ git branch -a
* branchxyz
  master
  remotes/origin/HEAD -> origin/master
  remotes/origin/branchxyz
  remotes/origin/branch123
```

You see, 'git clone git://example.com/myprojectt' fetches everything, even the branches, you just have to checkout them, then your local branch will be created.

share improve this answer follow

edited May 6 '13 at 16:22

Steven Penny
74.5k ● 45 ● 289 ● 333

answered May 31 '11 at 11:40

rapher
415 ● 4 ● 3

add a comment

25

You only need to use "git clone" to get all branches.

```
git clone <your_http_url>
```

Even though you only see master branch, you can use "git branch -a" to see all branches.

```
git branch -a
```

And you can switch to any branch which you already have.

```
git checkout <your_branch_name>
```

Don't worry that after you "git clone", you don't need to connect with the remote repo, "git branch -a" and "git checkout " can be run successfully when you close your wifi. So it is proved that when you do "git clone", it already has copied all branches from the remote repo. After that, you don't need the remote repo, your local already has all branches' codes.

Haimei
10.4k 3 42 33

Very clear response here. Lots of folks confused about this topic. – XMAN Oct 20 '18 at 23:12

I'd like to second your statement "can be run successfully when you close your wifi". "git clone" really does result in a repo containing all branches. – bvgheluwe Feb 28 '19 at 13:32

add a comment

24

A `git clone` is supposed to copy the entire repository. Try cloning it, and then run `git branch -a`. It should list all the branches. If then you want to switch to branch "foo" instead of "master", use `git checkout foo`.

share improve this answer follow

edited Oct 28 '17 at 15:33

answered Sep 15 '08 at 22:46

MattoxBeckman
3,654 2 16 16

1 You can run git commands with or without the hyphen. Both "git-branch" and "git branch" will work. – Peter Boughton Sep 15 '08 at 22:55

21 Maybe this answer was given a long time ago when git worked differently, but I think it's misleading today. `git clone` does download all the remote branches, but it only makes a local branch of master. Since `git branch` only shows local branches, you need `git branch -a` to see remote branches, too. – Cerran Mar 5 '14 at 13:05

Thank you. This is kind of strange default behavior IMO. I'll just chalk it up to more cryptic gitness. If it downloaded the branches, why would it hide them when calling git branch? – Adam Hughes Oct 26 '16 at 14:27

1 @Cerran, thanks; I've updated my answer accordingly. – MattoxBeckman Oct 28 '17 at 15:34

"does download all the remote branches, but it only makes a local branch of master". I need help understanding this. It seems git clone does NOT clone any branches except master, as when you do "git branch -a" it shows that the "develop" branch is only at "remotes/origin/develop". This must be saying that you dont have this branch anywhere locally, it only exists currently on the origin right? – John Little Jan 23 '18 at 10:47

add a comment

19

Use my tool git_remote_branch (you need Ruby installed on your machine). It's built specifically to make remote branch manipulations dead easy.

Each time it does an operation on your behalf, it prints it in red at the console. Over time, they finally stick into your brain :-)

If you don't want grb to run commands on your behalf, just use the 'explain' feature. The commands will be printed to your console instead of executed for you.

Finally, all commands have aliases, to make memorization easier.

Note that this is alpha software ;-)

Here's the help when you run grb help:

```
git_remote_branch version 0.2.6

  Usage:
```

```
grb publish branch_name [origin_server]

grb rename branch_name [origin_server]

grb delete branch_name [origin_server]

grb track branch_name [origin_server]



Notes:
- If origin_server is not specified, the name 'origin' is assumed
  (git's default)
- The rename functionality renames the current branch

The explain meta-command: you can also prepend any command with the
keyword 'explain'. Instead of executing the command, git_remote_branch
will simply output the list of commands you need to run to accomplish
that goal.

Example:
  grb explain create
  grb explain create my_branch github

All commands also have aliases:
create: create, new
delete: delete, destroy, kill, remove, rm
publish: publish, remotize
rename: rename, rn, mv, move
track: track, follow, grab, fetch
```

share improve this answer follow

answered Sep 20 '08 at 13:53

webmat
47.5k ● 12 ● 51 ● 58

6 Word to the wise: It looks like this project was abandoned around the time this answer was posted. I can't find any updates after 2008. Caveat emptor and all that. If I'm wrong, I hope someone will edit and provide a current pointer, because I'd love to have a tool like this handy. – bradheintz Apr 27 '11 at 21:53

@bradheintz check this answer, it sets up a git alias: stackoverflow.com/a/16563327/151841 – user151841 Apr 19 '19 at 15:06

add a comment

19

all the answers I saw here are valid but there is a much cleaner way to clone a repository and to pull all the branches at once.

When you clone a repository all the information of the branches is actually downloaded but the branches are hidden. With the command

```
$ git branch -a
```

you can show all the branches of the repository, and with the command

```
$ git checkout -b branchname origin/branchname
```

you can then "download" them manually one at a time.

However, when you want to clone a repo with a lot of branches all the ways illustrated are above are lengthy and tedious in respect to a much cleaner and quicker way that I am going to show, though it's a bit complicated. You need three steps to accomplish this:

1. First step

create a new empty folder on your machine and clone a mirror copy of the .git folder from the repository:

```
$ cd ~/Desktop && mkdir my_repo_folder && cd my_repo_folder
$ git clone --mirror https://github.com/planetoftheweb/responsivebootstrap.git .git
```

the local repository inside the folder my_repo_folder is still empty, there is just a hidden .git folder now that you can see with a "ls -alt" command from the terminal.

2. Second step

switch this repository from an empty (bare) repository to a regular repository by switching the boolean value "bare" of the git configurations to false:

```
$ git config --bool core.bare false
```

3. Third Step

Grab everything that inside the current folder and create all the branches on the local machine, therefore making this a normal repo.

```
$ git reset --hard
```

So now you can just type the command "git branch" and you can see that all the branches are downloaded.

This is the quick way in which you can clone a git repository with all the branches at once, but it's not something you wanna do for every single project in this way.

share improve this answer follow

answered Dec 6 '15 at 20:08

FedericoCapaldo
1,150 ● 15 ● 27

I don't like your usage of the word "download" in ... *"download" them manually one at a time*. All info is, in fact, already downloaded after having cloned the repo. The only thing one needs to do is creating local tracking branches (which is also possible when offline, which proves that all the information is in the repo). – bvgheluwe Feb 28 '19 at 13:35

@bvgheluwe that's why it is in quotes. – FedericoCapaldo Mar 5 '19 at 16:21

add a comment

15

Cloning from a local repo will not work with git clone & git fetch: a lot of branches/tags will remain unfetched.

To get a clone with all branches and tags.

```
git clone --mirror git://example.com/myproject myproject-local-bare-repo.git
```

```
git clone --mirror git://example.com/myproject myproject/.git
cd myproject
git config --unset core.bare
git config receive.denyCurrentBranch updateInstead
git checkout master
```

share improve this answer follow

answered Feb 23 '17 at 23:55

raisercostin
7,305 ● 3 ● 51 ● 62

add a comment

13

Looking at one of answers to the question I noticed that it's possible to shorten it:

```
for branch in  `git branch -r | grep -v 'HEAD\|master'`; do
 git branch --track ${branch##*/} $branch;
done
```

But beware, if one of remote branches is named as e.g. admin_master it won't get downloaded!

Thanks to bigfish for original idea

share improve this answer follow

answered Feb 19 '15 at 21:33

Tebe
2,784 ● 4 ● 31 ● 53

You can improve the regex, or maybe use Awk instead of `grep`, to improve the filter to avoid false positives. – tripleee Nov 4 '15 at 12:37

all the branches are 'origin\my_branch_name', which is not definitely what I want. – Знаđошfаχ Sep 1 '16 at 17:49

I've not seen the construct ${branch##*/} before - looks really useful - any idea where I can find out more on that? can't seem to find under bash anywhere. Thx. – SaminOz May 7 '17 at 11:44

add a comment

13

OK, when you clone your repo, you have all branches there...

If you just do `git branch`, they are kind of hidden...

So if you'd like to see all branches name, just simply add `--all` flag like this:

`git branch --all` or `git branch -a`

If you just checkout to the branch, you get all you need.

But how about if the branch created by someone else after you clone?

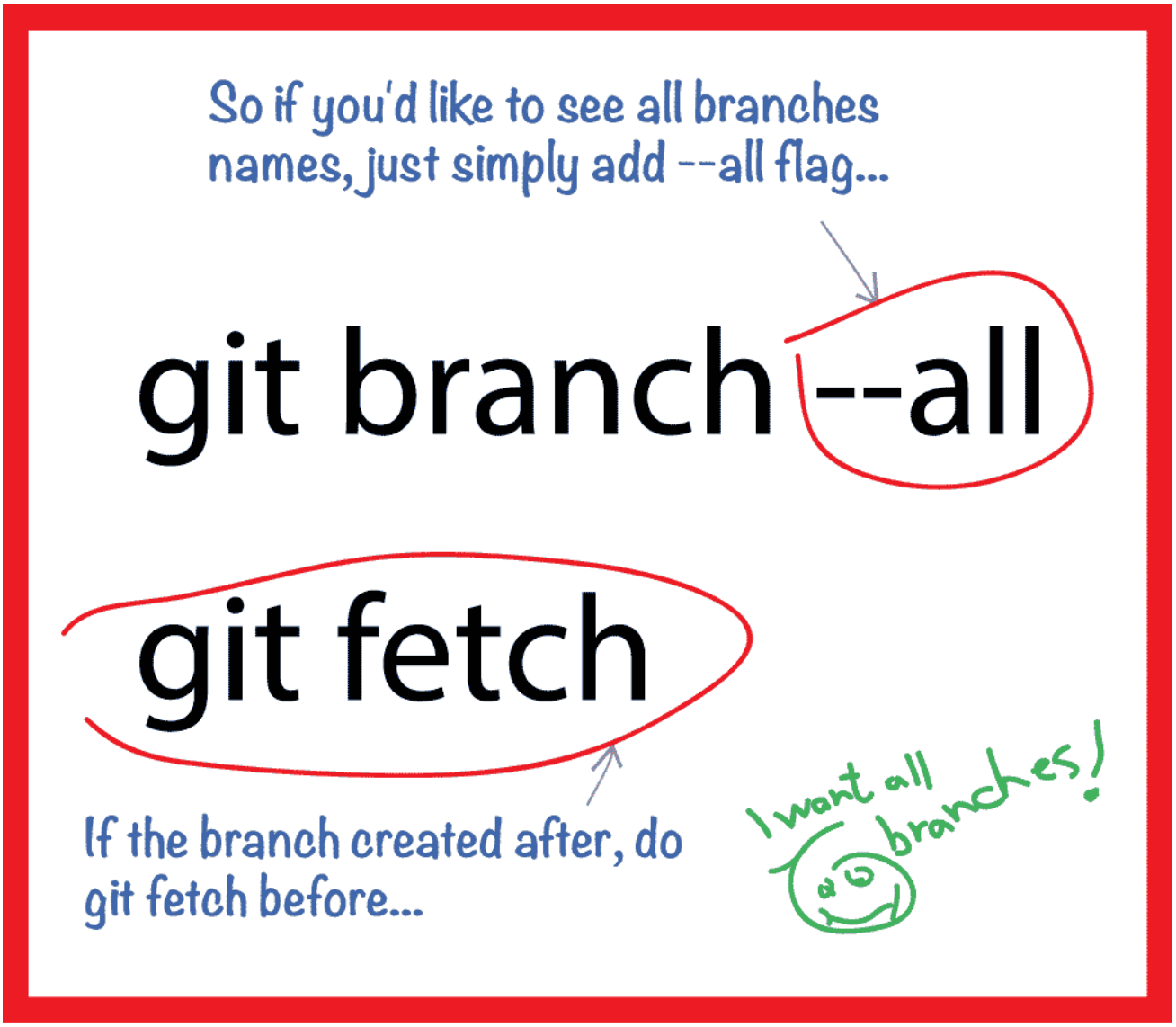
In this case, just do:

`git fetch`

and check all branches again...

If you like to fetch and checkout at the same time, you can do:

Also created the image below for you to simplify what I said:



share improve this answer follow

edited Mar 23 '18 at 4:05

answered Jul 22 '17 at 18:29

Alireza
73.9k ● 18 ● 223 ● 145

3 There is a difference between "you have it" and "you see it". git branch -all will NOT list the remote branches any more when you remove the remote repository. – Gustave Mar 1 '18 at 11:23

add a comment

12

```
#!/bin/bash
for branch in `git branch -a | grep remotes | grep -v HEAD | grep -v master `; do
   git branch --track ${branch#remotes/origin/} $branch
done
```

These code will pull all remote branches code to local repo.

1

share improve this answer follow

answered Dec 11 '16 at 2:20

ref. coderwall.com/p/0ypmka/git-clone-all-remote-branches-locally – arcseldon Apr 15 '17 at 15:13

add a comment

10

For copy-paste into command line:

```
git checkout master ; remote=origin ; for brname in `git branch -r | grep $remote | grep -v master | grep -v HEAD | awk '{gsu
```

For more readibility:

```
git checkout master ;
remote=origin ;
for brname in `
    git branch -r | grep $remote | grep -v master | grep -v HEAD
    | awk '{gsub(/^[^\/]+\//,"",$1); print $1}'
`; do
    git branch -D $brname ;
    git checkout -b $brname $remote/$brname ;
done ;
git checkout master
```

This will:

1. check out master (so that we can delete branch we are on)
2. select remote to checkout (change it to whatever remote you have)
3. loop through all branches of the remote except master and HEAD
    1. delete local branch (so that we can check out force-updated branches)
    2. check out branch from the remote
4. check out master (for the sake of it)

Based on answer of VonC.

share improve this answer follow

edited May 23 '17 at 11:47 Community ♦ 1 ● 1

answered Dec 20 '13 at 6:38 ikaruss 471 ● 4 ● 13

add a comment

10

None of these answers cut it, except user nobody is on the right track.

I was having trouble with moving a repo from one server/system to another. When I cloned the repo, it only created a local branch for master so when I pushed to the new remote, only master branch was pushed.

So I found these two methods VERY useful. Hope they help someone else.

Method 1:

```
git clone --mirror OLD_REPO_URL
cd new-cloned-project
mkdir .git
mv * .git
```

```
git push --all newrepo
git push --tags newrepo
```

Method 2:

```
git config --global alias.clone-branches '! git branch -a | sed -n "/\/HEAD /d; /\/master$/d; /remotes/p;" | xargs -L1 git c
git clone OLD_REPO_URL
cd new-cloned-project
git clone-branches
git remote add newrepo NEW_REPO_URL
git push --all newrepo
git push --tags newrepo
```

share improve this answer follow

answered Jan 23 '15 at 17:51

Gaui
7,148 ● 12 ● 58 ● 81

add a comment

10

I wrote this small Powershell functions to be able to checkout all my git branches, that are on origin remote.

```
Function git-GetAllRemoteBranches {
    iex "git branch -r"                     <# get all remote branches #> `
    | % { $_ -Match "origin\/(?'name'\S+)" }  <# select only names of the branches #> `
    | % { Out-Null; $matches['name'] }        <# write does names #>
}


Function git-CheckoutAllBranches {
   git-GetAllRemoteBranches `
       | % { iex "git checkout $_" }          <# execute ' git checkout <branch>' #>
}
```

More git functions can be found on my git settings repo

share improve this answer follow

answered Jun 19 '16 at 10:54

gringo_dave
992 ● 13 ● 21

add a comment

9

Here's an answer that uses awk. This method should suffice if used on a new repo.

```
git branch -r | awk -F/ '{ system("git checkout " $NF) }'
```

Existing branches will simply be checked out, or declared as already in it, but filters can be added to avoid the conflicts.

It can also be modified so it calls an explicit `git checkout -b <branch> -t <remote>/<branch>` command.

This answer follows Nikos C.'s idea.

Alternatively we can specify the remote branch instead. This is based on murphytalk's answer

```
git branch -r | awk '{ system("git checkout -t " $NF) }'
```

It throws fatal error messages on conflicts but I see them harmless.

Both commands can be aliased.

Using nobody's answer as reference, we can have the following commands to create the aliases:

```
git config --global alias.clone-branches '! git branch -r | awk -F/ "{ system(\"git checkout \" \$NF) }"'
git config --global alias.clone-branches '! git branch -r | awk "{ system(\"git checkout -t \" \$NF) }"'
```

Personally I'd use `track-all` or `track-all-branches`.

share improve this answer follow

answered Jun 20 '19 at 14:05

konsolebox
56.3k ● 7 ● 75 ● 89

Just wanted to thank you. This worked perfectly and it doesn't suffer from various problems related to cloning a bare repo such as some pull behaviour changing etc. – mahonya Dec 16 '19 at 16:27

add a comment

8

I needed to do exactly the same. Here is my Ruby script.

```
#!/usr/bin/env ruby

local = []
remote = {}

# Prepare
%x[git reset --hard HEAD]
%x[git checkout master] # Makes sure that * is on master.
%x[git branch -a].each_line do |line|
  line.strip!
  if /origin\//.match(line)
    remote[line.gsub(/origin\//, '')] = line
  else
    local << line
  end
end
# Update
remote.each_pair do |loc, rem|
  next if local.include?(loc)
  %x[git checkout --track -b #{loc} #{rem}]
end
%x[git fetch]
```

share improve this answer follow

edited Jun 16 '13 at 13:45

Peter Mortensen
26.3k ● 21 ● 91 ● 121

answered Dec 10 '10 at 23:12

user43685
1,296 ● 3 ● 15 ● 21

See answer I posted further down to avoid having to run this script at all. – lacostenycoder Dec 8 '18 at 15:41

`git clone --mirror` on the original repo works well for this.

8

```
git clone --mirror /path/to/original.git
git remote set-url origin /path/to/new-repo.git
git push -u origin
```

share improve this answer follow

answered May 25 '18 at 17:34

Bernd Jungblut
191 ●1 ●5

add a comment

8

1

The accepted answer of `git branch -a` only *shows* the remote branches. If you attempt to `checkout` the branches you'll be unable to unless you still have network access to the origin server.

If you're looking for a *self-contained clone or backup* that *includes* all remote branches and commit logs, use:

```
git clone http://user@repo.url
git pull --all
```

Credit: Gabe Kopley's for suggesting using `git pull --all`.

Note:
Of course, if you no longer have network access to the `remote/origin` server, `remote/origin branches` will not have any updates reflected in them. Their revisions will reflect commits from the date and time you performed the 2 commands above.

Checkout a *local* branch in the usual way with `git checkout remote/origin/` Use `git branch -a` to reveal the remote branches saved within your `clone` repository.
To checkout ALL your *clone* branches to *local branches* with one command, use one of the bash commands below:

```
$ for i in $(git branch -a |grep 'remotes' | awk -F/ '{print $3}' \
| grep -v 'HEAD ->');do git checkout -b $i --track origin/$i; done
```

OR

If your repo has nested branches then this command will take that into account:

```
for i in $(git branch -a |grep 'remotes' |grep -v 'HEAD ->');do \
basename ${i##\./} | xargs -I {} git checkout -b {} --track origin/{}; done
```

The above commands will `checkout` a local branch into your local git repository, named the same as the *`remote/origin/`* and set it to `--track` changes from the remote branch on the *`remote/origin`* server should you regain network access to your origin repo server once more and perform a `git pull` command in the usual way.

share improve this answer follow

edited Jun 25 at 3:28

answered Sep 25 '19 at 15:03

Tony Barganski

This is what I came her for! – Blake Yarbrough 5 hours ago

add a comment

7

Git usually (when not specified) fetches all branches and/or tags (refs, see: `git ls-refs`) from one or more other repositories along with the objects necessary to complete their histories. In other words it fetches the objects which are reachable by the objects that are already downloaded. See: What does `git fetch` really do?

Sometimes you may have branches/tags which aren't directly connected to the current one, so `git pull --all` / `git fetch --all` won't help in that case, but you can list them by:

```
git ls-remote -h -t origin
```

and fetch them manually by knowing the ref names.

So to fetch them all, try:

```
git fetch origin --depth=10000 $(git ls-remote -h -t origin)
```

The `--depth=10000` parameter may help if you've shallowed repository.

Then check all your branches again:

```
git branch -avv
```

If above won't help, you need to add missing branches manually to the tracked list (as they got lost somehow):

```
$ git remote -v show origin
...
  Remote branches:
    master      tracked
```

by `git remote set-branches` like:

```
git remote set-branches --add origin missing_branch
```

so it may appear under `remotes/origin` after fetch:

```
$ git remote -v show origin
...
  Remote branches:
    missing_branch new (next fetch will store in remotes/origin)
$ git fetch
From github.com:Foo/Bar
 * [new branch]      missing_branch -> origin/missing_branch
```

Troubleshooting