The most confusing git terminology

n.b. This blog post dates from 2012, so some of it may be out of date now.

To add my usual disclaimer to the start of these blog posts, I should say that I love git; I think it's a beautiful and elegant system, and it saves me huge amounts of time in my daily work. However, I think it's a fair criticism of the system that its terminology is very confusing for newcomers, and in particular those who have come from using CVS or Subversion.

This is a personal list of some of my "favourite" points of confusion, which I've seen arise time and time again, both in real life and when answering questions on Stack Overflow. To be fair to all the excellent people who have contributed to git's development, in most cases it's clear that they are well aware that these terms can be problematic, and are trying to improve the situation subject to compatibility constraints. The problems that seem most bizarre are those that reuse CVS and Subversion terms for completely different concepts – I speculate a bit about that at the bottom.

"update"

If you've used Subversion or CVS, you're probably used to "update" being a command that goes to the remote repository, and incorporates changes from the remote version into your local copy – this is (very broadly) analogous to "git pull". So, when you see the following error message when using git:

```
foo.c: needs update
```

You might imagine that this means you need to run "git pull". However, that's wrong. In fact, what "needs update" means is approximately: "there are local modifications to this file, which you should probably commit or stash".

"track" and "tracking"

The word "track" is used in git in three senses that I'm aware of. This ambiguity is particularly nasty, because the latter two collide at a point in learning the system where newcomers to git are likely to be baffled *anyway*. Fortunately, this seems to have been recognized by git's developers (see below).

1. "track" as in "untracked files"

To say that a file is tracked in the repository appears to mean that it is either present in the index or exists in the commit pointed to by HEAD. You see this usage most often in the output of "git status", where it will list "untracked files":

```
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#    .classpath
```

This sense is relatively intuitive, I think – it was only after complaining for a while about the next two senses of "track" that I even remembered that there was also this one :)

2. "track" as in "remote-tracking branch"

As a bit of background, you can think of a *remote-tracking branch* as a local cache of the state of a branch in a remote repository. The most commonly seen example is `origin/master`, or, to name that ref in full, `refs/remotes/origin/master`. Such branches are usually updated by `git fetch` (and thus also potentially by `git pull`). They are also updated by a successful push to the branch in the remote repository that they correspond to.   You can merge from them, examine their history, etc. but you can't work directly on them.

The sense of "track" in the phrase "remote-tracking branch" is indicating that the remote-tracking branch is tracking the state of the branch in the remote repository the last time that remote-tracking branch was updated. So, you might say that `refs/remotes/origin/master` is tracking the state of the branch `master` in `origin`.

The "tracking" here is defined by the refspec in the config variable `remote.<remote-name>.fetch` and the URL in the config variable `remote.<remote-name>.url`.

3. "track" as in "git branch –track foo origin/bar" and "Branch foo set up to track remote branch bar from origin"

Again, if you want to do some work on a branch from a remote repository, but want to keep your work separate from everything else in your repository, you'll typically use a command like the following (or one of its *many* "Do What I Mean" equivalents):

```
git checkout --track -b foo origin/bar
```

… which will result in the following messages:

```
Branch foo set up to track remote branch bar from origin
Switched to a new branch 'foo'
```

The sense of "track" both in the command and the output is distinct from the previous sense – it means that config options have been set that associate your new local branch with another branch in the remote repository. The documentation sometimes refers to this relationship as making `bar` in `origin` "upstream" of `foo`. This "upstream" association is very useful, in fact: it enables nice features like being able to just type `git pull` while you're on branch `foo` in order to fetch from `origin` and then merge from `origin/bar`. It's also how you get helpful messages about the state of your branch relative to the remote-tracking branch, like "Your branch `foo` is 24 commits ahead of `origin/bar` and can be fast-forwarded".

The tracking here is defined by config variables `branch.<branch-name>.remote` and `branch.<branch-name>.merge`.

"tracking" Summary

Fortunately, the third sense of “tracking” seems to be being carefully deprecated – for example, one of the possible options for `push.default` used to be `tracking`, but this is now deprecated in favour of the option name `upstream`. The commit message for 53c403116 says:

push.default: Rename ‘tracking’ to ‘upstream’

Users are sometimes confused with two different types of “tracking” behavior in Git: “remote-tracking” branches (e.g. refs/remotes//*) versus the merge/rebase relationship between a local branch and its @{upstream} (controlled by branch.foo.remote and branch.foo.merge config settings).*

When the push.default is set to ‘tracking’, it specifies that a branch should be pushed to its @{upstream} branch. In other words, setting push.default to ‘tracking’ applies only to the latter of the above two types of “tracking” behavior.

In order to make this more understandable to the user, we rename the push.default == ‘tracking’ option to push.default == ‘upstream’.

push.default == ‘tracking’ is left as a deprecated synonym for ‘upstream’.

“commit”

In CVS and Subversion, “commit” means to send your changes to the remote repository. In git the action of committing (with “git commit”) is entirely local; the closest equivalent of “cvs commit” is “git push”. In addition, the word “commit” in git is used as both a verb and a noun (although frankly I’ve never found this confusing myself – when you commit, you create a commit).

“checkout”

In CVS and Subversion “checkout” creates a new local copy of the source code that is linked to that repository. The closest command in git is “git clone”. However, in git, “git checkout” is used for something completely distinct. In fact, it has two largely distinct modes of operation:

1. To switch HEAD to point to a new branch or commit, in the usage `git checkout`

`<branch>`. If `<branch>` is genuinely a local branch, this will switch to that branch (i.e. HEAD will point to the ref name) or if it otherwise resolves to a commit will detach HEAD and point it directly to the commit's object name.
2. To replace a file or multiple files in the working copy and the index with their content from a particular commit or the index. This is seen in the usages: `git checkout --` (update from the index) and `git checkout <tree-ish> --` (where `<tree-ish>` is typically a commit).

(`git checkout` is also frequently used with `-b`, to create a new branch, but that's really a sub-case of usage 1.)

In my ideal world, these two modes of operation would have different verbs, and neither of them would be "checkout".

Update in 2023: This has now happened – you can now use "git switch" for the former cases and "git restore" for the latter. This is a very welcome change :)

"HEAD" and "head"

There are usually many "heads" (lower-case) in a git repository – the tip of each branch is a head. However, there is only one `HEAD` (upper-case) which is a symbolic ref which points to the current branch or commit.

"fetch" and "pull"

I wasn't aware of this until Roy Badami pointed it out, but it seems that git and Mercurial have opposite meanings for "fetch" and "pull" – see the top two lines in this table of git / hg equivalences. I think it's understandable that since git's and Mercurial's development were more or less concurrent, such unfortunate clashes in terminology might occur.

"push" and "pull"

"git pull" is not the opposite of "git push"; the closest there is to an opposite of "git push" is "git fetch".

“hash”, “SHA1”, “SHA1sum”, “object name” and “object identifier”

These terms are often used synonymously to mean the 40 characters hexadecimal strings that uniquely identify *objects* in git. “object name” seems to be the most official, but the least used in general. Referring to an object name as a SHA1sum is potentially confusing, since the object name for a blob is not the same as the SHA1sum of the file.

“remote branch”

This term is only used occasionally in the git documentation, but it’s one that I would always try to avoid because it tends to be unclear whether you mean “a branch in a remote repository” or “a remote-tracking branch”. Whenever a git beginner uses this phrase, I think it’s worth clarifying this, since it can avoid later confusion.

“index”, “staging area” and “cache”

As nouns, these are all synonyms, which all exist for historical reasons. Personally, I like “staging area” the best since it seems to be the easiest concept to understand for git beginners, but the other two are used more commonly in the documentation.

When used as command options, `--index` and `--cached` have distinct and consistent meanings, as explained by Junio C. Hamano in this useful blog post.

Why are there so many of these points of confusion?

I would speculate that the most significant effect that contributed to these terminology confusions is that git was being actively used by an enthusiastic community from *very* early in its development, which means that early names for concepts have tended to persist for the sake of compatibility and consistency. That doesn’t necessarily account for the many conflicts with CVS / Subversion usage, however.

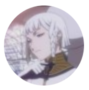
To be fair to git, thinking up verbs for particular commands in any software is tough, and there have been enough version control systems written that to completely avoid clashes would lead to some convoluted choices. However, it's hard to see git's use of CVS / Subversion terminology for completely different concepts as anything but perverse. Linus has made it very clear many times that he hated CVS, and joked that a design principle for git was WWCVSND (What Would CVS Never Do); I'm sympathetic to that, as I'm sure most are, especially after having switched to the DVCS mindset. However, could that attitude have extended to deliberately disregarding concerns about terminology that might make it actively harder for people to migrate to git from CVS / Subversion? I don't know nearly enough about the early development of git to know. However, it wouldn't have been tough to find better choices for `commit`, `checkout` and `update` in each of their various senses.

Posted 2012-05-07 in git
by mark

Tags:

Comments

21 responses to "The most confusing git terminology"



Dmitriy Matrosov
2012-06-09

Hi, Mark.

Thanks for yor excellent explanation of “tracking” branches! This is exactly what i searched for.

Also, i think, here is typo:
You wrote “The documentation sometimes refers to this relationship as making foo in origin “upstream” of bar.”, but it should be “bar in origin upstream of foo”.

Reply

mark
2012-06-11

Hi Dimitriy: thanks for your kind comment and the correction. I’ve edited the post to fix that now.

Reply

harold
2012-10-05

Really good article – a lot of these terms are thrown about in git documentation/tutorials without much explanation and can certainly trip you up.

Reply

ahmet
2012-11-26

“Why are there so many of these points of confusion?”

Probably beacuse Linus is a dictator and he probaby didn’t give shit about others’ opinions and past works for the terminology.

Reply

Bri
2014-08-14

Right! He forgot to include the input of someone who knows an inkling about user-friendliness. Imagine how much easier GIT SWITCH would be to remember than GIT CHECKOUT. The whole system was made unnecessarily complicated just by having awfully named functions.

Reply

Alberto Fonseca
2018-03-22

Good point. Another reason is probably the agile paradigm that tells us that only “working code” is important, so developers happily cast aside anything even remotely resembling logical, structured, conceptional work, which works only based on a precisely defined set of domain vocabulary (which is exactly what we’re missing here).

Reply

Jennifer
2013-02-05

I find the term *upstream* confusing. You mention one usage in this article. However, GitHub help recommends *To keep track of the original repo [you forked from], you need to add* *another remote named upstream*. So when someone says to push *upstream*, it's ambiguous. I'm not sure if this usage is typical of Git in general or just GitHub, but it sure left me confused for a while.

Reply

Ian
2013-11-21

"remote branch"

This term is only used occasionally in the git documentation, but it's one that I would always try to avoid because it tends to be unclear whether you mean "a branch in a remote repository" or "a remote-tracking branch". Whenever a git beginner uses this phrase, I think it's worth clarifying this, since it can avoid later confusion.

Doh! That's exactly what I'm trying to find out, but then you don't actually give us the answer!

Reply

admin
2013-11-21

Hi Ian, That's the thing – you often can't tell without context. Where did you see "remote branch" used, or in what context? ("a branch in a remote repository" is the more accurate interpretation, I think, but people who don't understand remote-tracking branches might use the term differently.)

Reply

Oleh
2014-05-28

Thanks for the helpful git articles. I'm a git noob, and the more different sources I read, the closer my understanding converges on what's really going on. It's just a shame that numerous git commands have ended up being labels where one must simply memorize what they really do — might as well have called them things like "abc" and "xyz" instead of terms that initially lead one astray.

If you're motivated to write another one, here's a suggestion. Maybe you could review all the places that git stores stuff, like working repository, index (aka cache or staging area), remote repository, local copy of remote repository (remote tracking branch?), stash (and stash versions), etc., and list the commands that transfer data in various ways among these places. I've had to gradually formulate and revise these ideas in my own mind as I've been reading about git, and if I had an up-front explanation of this at the outset, it would've saved me lots of time.

Reply

mark
2014-06-03

Thanks for the kind words and your suggestion Oleh. I'll certainly consider it – there are quite a few good diagrammatic representations of the commands that change what's stored in the index, working tree, etc., but maybe there's something more to do. (You can find some good examples with an image search for `git index diagram`.)

Reply

bryan chance
2017-12-06

Oleh, That would be a great write up. I think those are the rest of the most confusing git terminoloy. :p I'm new to git as well and I thought I was just stupid because I couldn't get a handle on it. This is 2017 and yes git terminologies still baffle users.

Reply

Savanna
2018-11-13

I've been using git for years and I'm only just now diving into how it actually works and what the commands actually mean since I'm working at a place that has a rather complicated branching structure… in the past I simply worked on master since I was the only or one of few developers. git pull and git push were usually enough. Now I find myself often totally confused and realized I don't actually know what the heck I'm doing with git, so I'm reading all these articles and watching videos. It's been too long coming! Haha

Reply

Alistair
2014-11-12

> The sense of "track" in the phrase "remote-tracking branch" is indicating that the remote-tracking branch is tracking the state of the branch in the remote repository the last time that remote-tracking branch was updated.

In the light of your paragraph about "update", maybe "fetched" would be a better word than "updated"?

> 3. "track" as in "git branch –track foo origin/bar"

I don't understand how senses 2 and 3 are different. Specifically, sense 2 seems to be the special case of 3 where "foo" and "bar" coincide. I.e. the "upstream" of a remote-tracking branch is the branch of the same name on "origin". Is that right?

> commit
>
> As a noun

I agree. This terminology is fine, and not really confusing at all. The closest english word would be “commitment”, I think, but in the context it is good to have a slightly different word.

> “fetch” and “pull” […] “push” and “pull”

This juxtaposition is hilarious. In the former you are very diplomatic in suggesting that git’s version is an arbitrary choice, made by historical accident, but then in the latter you immediately make it look like exactly the opposite choice would be better.

Reply

admin
2014-12-30

> In the light of your paragraph about “update”, maybe “fetched” would be a better word than “updated”

I see what you mean, but I think it would probably be more confusing to use “fetched”; the colloquial sense of update is what I mean and the “needs update” sense in git is surprising and wouldn’t make sense in this context anyway.

> I don’t understand how senses 2 and 3 are different. Specifically, sense 2 seems to be the special case of 3 where “foo” and “bar” coincide. I.e. the “upstream” of a remote-tracking branch is the branch of the same name on “origin”. Is that right?

In sense 2 (“remote-tracking branch”) the tracking can only be about this special type of branch tracking the branch in a remote repository; very often there’s no corresponding “local” branch (as opposed to a remote-tracking branch). In sense 3 (“–track”) there *must* be a local branch, and the tracking is describing an association between the local branch and one in the remote repository (and usually to a corresponding remote-tracking branch).

Reply

Philip A
2015-04-01

Thanks so much for this really useful and informative blog post which I have only recently discovered. I think focusing in on the confusing aspects of Git is a great way to improve one's general understanding. Particularly helpful are your explanations of 'remote tracking branches' compared with 'branches which track branches on a remote'. One very minor thing which confused me was: "Your branch foo is 24 commits ahead of origin/bar and can be fast-forwarded". Shouldn't it be 'behind' instead of 'ahead' ? Apologies if I have misunderstood. Anyway, once again … really good post :-)

Reply

Matthew Astley
2016-02-17

Thanks – good list, and several of us here misuse the jargon.

When describing the state of the work tree: clean, up-to-date and unmodified are easily mixed up.

Going by the command name, clean should mean no untracked files (aka. cruft) in the work tree.

Unmodified should mean no changes to tracked files, but the command to do this is reset – another naming clash?

Up-to-date is trickier to define. There may be copies elsewhere containing commits you can't yet see. Is the HEAD (local branch) up-to-date with one remote at a point in time when the fetch ran? Should all local branches be up-to-date?

Reply

Mike Weilgart
2016-04-16

This is an excellent article; thank you!

Particularly helpful was the explanation of the three senses in which the word “track” may be used.

The light went on when I realized that there are actually *three* things going on when you deal with a remote, not just two:

1. The branch on the remote repository;
2. The remote tracking branch in your local repository;
3. The local branch associated with the remote tracking branch.

This clarified for me how I should go about instructing students in this aspect of Git; thank you!

Reply

Simon Bagley
2016-06-02

Thank you for the very informative article. I am a complete novice with Git, and your article will help me understand the many vary confusing names of commands, options etc.
It would be useful if you defined the terms ‘refs’, and ‘refspec’ before you used them.

Reply

Doug Kimzey
2019-09-06

I could not agree more. I have been using git for about 3 months now. Git is not a full blown language but a source control tool.

The ambiguity of the git syntax contributes greatly to the confusion experienced by developers facing git for the first time. The concepts are not difficult. The ambiguity of the git syntax constantly forces you to reach for a cheat sheet. The commands do not clearly describe their corresponding actions. Source control is a crucial part of any software project. Source control is not an area that should be managed by confusing and ambiguous terminology and syntax. Confusion puts work at risk and adds time to projects.

Commands that concisely describe actions taken with source control are very important.

Thousands of developers use git. Millions of Americans use IRS Tax Forms. This does not mean that Millions of Americans find IRS Tax Forms clear and easy to understand. Git terminology unnecessarily costs time and effort in translation.

Reply

Doug Kimzey
2023-04-11

This is very well said. If the git syntax was concise:

– There would be far fewer books on amazon.
– The number of up votes on git questions on stackoverflow would be much smaller (some of these are literally in the thousands).
– There would be fewer cheat sheets and diagrams.
– The investment in time and effort would be much less.

The git documentation is poor because one vague command is described in terms and several ambiguous and confusing commands.

Grammar is not the only measure of the quality of documentation.

I am adopting some of the rules in the Simplified Technical English standard (ASD-STE100) to the naming of commands used in console applications and utilities.

The goals of this standard are:

– Reduce ambiguity.
– Improve clarity of the technical text.
– Make user manuals more comprehensive for non-native speakers of English.
– Create better conditions for both human and machine translation.

These goals should be carried to command syntax.

Reply

Leave a Reply

Your email address will not be published. Required fields are marked *

Comment *

Name *

Email *

Website

Post Comment

This site uses Akismet to reduce spam. Learn how your comment data is processed.