Git tips from the trenches (/git-tips-from-the-trenches/)

FEBRUARY 1, 2014 BY CSABA OKRONA (HTTPS://PLUS.GOOGLE.COM/104319029501878849601/?REL=AUTHOR)

Facebook Twitter Google+

After a few years with git everyone has their own bag o' tricks - a collection of bash aliases, one liners and habits that make his daily work easier.

I've gathered a handful of these with varying complexity hoping that it can give a boost to you. I will not cover git or VCS basics at all, I'm assuming you're already a git-addict.

So fire up your favorite text editor and bear with me.

Check which branches are merged

After a while if you branch a lot you'll se your git branch -a output is polluted like hell (if you havent't cleaned up). It all the more true if you're in a team. So, from time to time you'll do the Big Spring Cleaning only to find it hard to remember which branch you can delete and which you shouldn't. Well, just check out your mainline branch (usually master) and:

```
$ git checkout master
$ git branch --merged
```

to see all the branches that have already been merged to the current branch (master in this case).

You can do the opposite of course:

```
$ git branch --no-merged
```

How about deleting those obsolete branches right away?

```
$ git branch --merged | xargs git branch -d
```

Alternative: use GitHub's Pull request UI if you've been a good sport and always used pull requests.

Find something in your entire git history

Sometimes you find yourself in the situation that you're looking for a specific line of code that you don't find with plain old grep - maybe someone deleted or changed it with a commit. You remember some parts of it but have no idea where and when you committed it. Fortunately git has your back on this. Let's fetch all commits ever then use git's internal grep subcommand to look for your string:

```
$ git rev-list --all | xargs git grep '<YOUR REGEX>'
$ git rev-list --all | xargs git grep -F '<YOUR STRING>'  # if you don't want to use regex
```

Fetch a file from another branch without changing your current branch

Local cherry-picking. Gotta love it. Imagine you're experimenting on your current branch and you suddenly realise you need a file from the oh-so-distant branch. What do you do? Yeah, you can stash, git checkout, etc. but there's an easier way to merge a single file in your current branch from another:

```
$ git checkout <OTHER_BRANCH> -- path/to/file
```

See which branches had the latest commits

Could also be useful for a spring cleaning - checking how 'old' those yet unmerged branches are. Let's find out which branch hadn't been committed to in the last decade. Git has a nice subcommand, 'for-each-ref' which can print information for each ref (duh) - the thing is that you can both customize the output format and sort!

```
$ git for-each-ref --sort=-committerdate --format='%(refname:short) %(committerdate:short)'
```

It will output branches and tags, too.

This deserves an alias, don't you think?

```
$ git config --global alias.springcleaning "for-each-ref --sort=-committerdate --format='%(refname:short) %(co
mmitterdate:short)'"
```

Making typos?

Git can autocorrect you.

```
$ git config --global help.autocorrect 1
$ git dffi
WARNING: You called a Git command named 'dffi', which does not exist.
Continuing under the assumption that you meant 'diff'
in 0.1 seconds automatically...
```

Autocomplete, anyone?

If you download this (https://raw.github.com/git/git/master/contrib/completion /git-completion.bash) file and modify your .bash_profile by adding:

```
source ~/.git-completion.bash
```

Git will now autocomplete your partial command if you press TAB. Neat.

Hate remnant whitespace?

Let git strip it for you. Use the mighty .gitattributes file in the root of your project and say in it:

```
* filter=stripWhitespace
```

Or say you don't want this for all files (*), only scala sources:

```
*.scala filter=stripWhitespace
```

But the filter is not defined yet, so chop-chop:

```
$ git config filter.stripWhitespace.clean strip_whitespace
```

(Actually there are two types of filters: clean and smudge. Clean runs right before pushing, smudge is run right after pulling)

We still have to define what strip_whitespace is, so create a script on your PATH and of course make it executable:

```
#!/usr/bin/env ruby
STDIN.readlines.each do |line|
  puts line.rstrip
end
```

You could also do this as a precommit hook, of course.

Recovering lost data

The rule of thumb is that if you lost data but committed/pushed it somewhere, you're probably able to recover it. There are basically two ways:

reflog

Any change you make that affects a branch is recorded in the reflog. See:

```
$ git log -g
commit be5de4244c1ef863e454e3fb7765c7e0559a6938
Reflog: HEAD@{0} (Csaba Okrona <xxx@xx.xx>)
Reflog message: checkout: moving from master to master
Author: Robin Ward <xxx@xx.xx>
Date:   Fri Nov 8 15:05:14 2013 -0500

    FIX: Pinned posts were not displaying at the top of categories.
```

If you see your lost commit(s) there, just do a simple:

```
$ git branch my_lost_data [SHA-1]
```

Where SHA-1 is the hash after the 'commit' part. Now merge your lost data into your current branch:

```
$ git merge my_lost_data
```

git-fsck

```
$ git fsck --full
```

This gives you all the objects that aren't referenced by any other object (orphans). You can fetch the SHA-1 hash and do the same dance as above.

A nicer, one-line log

Get a color-coded, one-line-per-commit log showing branches and tags:

```
$ git log --oneline --decorate
355459b Fix more typos in server locale
b95e74b Merge pull request #1627 from awesomerobot/master
40aa62f adding highlight & fade to linked topic
15c29fd (tag: v0.9.7.3, tag: latest-release) Version bump to v0.9.7.3
c753a3c We shouldn't be matching on the `created_at` field. Causes tests to randomly fail.
dbd2332 Public user profile page shows if the user is suspended and why.
```

Highlight word changes in diff

Bored of the full-line highlights? This only highlights the changed words, nicely inline. Try:

```
$ git diff --word-diff
```

A shorter, pro git status

Showing only the important things.

```
$ git status -sb
## master...origin/master
?? _posts/2014-02-01-git-tips-from-the-trenches.md
?? images/git-beginner-share.png
?? images/git-beginner.jpg
```

Bored of setting up tracking branches by hand?

Make git do this by default:

```
$ git config --global push.default tracking
```

This sets up the link to the remote if it exists with the same branch name when you push.

Pull with rebase, not merge

To avoid those nasty merge commits all around.

```
$ git pull --rebase
```

Or do it automatically for any branch you'd like:

```
$ git config branch.master.rebase true
```

Or for all branches:

```
$ git config --global branch.autosetuprebase always
```

Find out which branch has a specific change

```
$ git branch --contains [SHA-1]
```

If you want to include remote tracking branches, add '-a'

Check which changes from a branch are already upstream

```
$ git cherry -v master
```

Show the last commit with matching message

```
$ git show :/regex
```

Write notes for commits

```
$ git notes add
```

You can share them by pushing - for more see http://git-scm.com/blog/2010/08/25/notes.html (http://git-scm.com/blog/2010/08/25/notes.html)

More cautious git blame

Before you play the blame game, make sure you check you're right with:

```
$ git blame -w  # ignores white space
$ git blame -M  # ignores moving text
$ git blame -C  # ignores moving text into other files
```

Aliases make life easier

These go to the '[alias]' section of your .gitconfig

```
ds = diff --staged      # git ds - diff your staged changes == review before committing.
st = status -sb         # smarter status - include tag and branch info
fup = log --since '1 day ago' --oneline --author <YOUR_EMAIL>  # I know what you did yesterday - great for fol
low-ups
ls = log --pretty=format:"%C(yellow)%h %C(blue)%ad%C(red)%d %C(reset)%s%C(green) [%cn]" --decorate --date=shor
t  # pretty one-line log with tags, branches and authors
lsv = log --pretty=format:"%C(yellow)%h %C(blue)%ad%C(red)%d %C(reset)%s%C(green) [%cn]" --decorate --date=sho
rt --numstat    # a verbose ls, shows changed files too

# some resets without explanation
r = reset
r1 = reset HEAD^
r2 = reset HEAD^^
rh = reset --hard
rh1 = reset HEAD^ --hard
rh2 = reset HEAD^^ --hard

# basic shortcuts
cp = cherry-pick
cl = clone
ci = commit
co = checkout
br = branch
diff = diff --word-diff
dc = diff --cached

# stash shortcuts
sl = stash list
sa = stash apply
ss = stash save

# log related - thanks to @mwd410
l = log
lh = log --graph
la = !git lh --date-order --all 2> /dev/null
lb = log --graph --simplify-by-decoration
lba = !git lb --all
h = !git --no-pager log --graph -n 15
a = !git --no-pager la -n 15
```

Did I miss something? Tell me in comments ;)

Facebook Twitter Google+

31 Comments Ochronus online Login

Sort by Best Share Favorite

Join the discussion…

rafaeldff • 22 days ago
You can search for changes over all commits with `git log -S<searchstring>`. Easier and probably faster than

`git rev-list --all | xargs git grep`</searchstring>
16 • Reply • Share ›

nobody in particular • 22 days ago
you have 'st' defined twice in your alias list
6 • Reply • Share ›

Hans • 21 days ago
in this alias list, ls & lsv should include single quote " ' ", instead of double quote. ". Otherwise, git show errors as "ambiguous argument '%C(blue)%ad%C(red)%d': unknown revision or path not in the working tree."
4 • Reply • Share ›

Joe Nelson • 22 days ago
Another one I love is to see the branches ordered by last commit time:

brs = for-each-ref --sort=-committerdate --format="%(committerdate:relative)%09%(refname:short)" refs/heads
4 • Reply • Share ›

joelparkerhenderson • 22 days ago
Thanks! Your git tips are great. For more tips and a large list of git alias commands: https://github.com/SixArm/sixa...
3 • Reply • Share ›

Ochronus Mod → joelparkerhenderson • 21 days ago
Wow, great collection!! Impressive, thanks for sharing
• Reply • Share ›

Yawar Amin • 22 days ago
You can push and pull notes just like other commits, see the Pro Git book :-)

Other posts you may like

A Docker primer – from zero to a running Django app (/docker-primer-django/)
A Rubyist's confessions on Python (/a-rubyists-confessions-on-python/)
Git bisect - debug with git (/git-bisect-debug-with-git/)
How to Steal a Facebook Identity (/steal-facebook-identity/)
Confessions of a knowledge junkie (/confessions-of-a-knowledge-junkie/)
Learn Clojure with Project Euler (/learn-clojure-with-project-euler/)
Tracking without cookies (/tracking-without-cookies/)
Top Technology and Software Trends – 2012 – 2013 (/top-technology-software-trends-2012-2013/)
Top 5 Trends and Technologies in Software Development (/top-5-trends-in-software-development/)
User tracking with HTTP Redirect (/user-tracking-http-redirect/)