LOG IN

6 handy Bash scripts for Git

These six Bash scripts will make your life easier when you're working with Git repositories.

By Bob Peterson

January 15, 2020 | 2 Comments | 13 min read 371 readers like this.

Image by: Opensource.com

More on Git

What is Git?

Git cheat sheet

Markdown cheat sheet

New Git articles

I wrote a bunch of Bash scripts that make my life easier when I'm working with Git repositories. Many of my colleagues say there's no need; that everything I need to do can be done with Git commands. While that may be true, I find the scripts infinitely more convenient than trying to figure out the appropriate Git command to do what I want.

1. gitlog

gitlog prints an abbreviated list of current patches against the master version. It prints them from oldest to newest and shows the author and description, with **H** for **HEAD**, **^** for **HEAD^**, **2** for **HEAD~2,** and so forth. For example:

```
$ gitlog
-----------------------[ recovery25 ]-----------------------
(snip)
11 340d27a33895 Bob Peterson     gfs2: drain the ail2 list afte
10 9b3c4e6efb10 Bob Peterson     gfs2: clean up iopen glock mes
 9 d2e8c22be39b Bob Peterson     gfs2: Do proper error checking
 8 9563e31f8bfd Christoph Hellwig gfs2: use page_offset in gfs2
 7 ebac7a38036c Christoph Hellwig gfs2: don't use buffer_heads
 6 f703a3c27874 Andreas Gruenbacher gfs2: Improve mmap write vs
 5 a3e86d2ef30e Andreas Gruenbacher gfs2: Multi-block allocatio
 4 da3c604755b0 Andreas Gruenbacher gfs2: Fix end-of-file handl
 3 4525c2f5b46f Bob Peterson     Rafael Aquini's slab instrumen
 2 a06a5b7dea02 Bob Peterson     GFS2: Add go_get_holdtime to g
 ^ 8ba93c796d5c Bob Peterson     gfs2: introduce new function r
 H e8b5ff851bb9 Bob Peterson     gfs2: Allow rgrps to have a mi
```

If I want to see what patches are on a different branch, I can specify an alternate branch:

```
$ gitlog recovery24
```

2. gitlog.id

gitlog.id just prints the patch SHA1 IDs:

```
$ gitlog.id
-----------------------[ recovery25 ]-----------------------
56908eeb6940 2ca4a6b628a1 fc64ad5d99fe 02031a00a251 f6f38da7dd1
```

Again, it assumes the current branch, but I can specify a different branch if I want.

3. gitlog.id2

gitlog.id2 is the same as **gitlog.id** but without the branch line at the top. This is handy for cherry-picking all patches from one branch to the current branch:

```
$ # create a new branch
$ git branch --track origin/master
$ # check out the new branch I just created
$ git checkout recovery26
$ # cherry-pick all patches from the old branch to the new one
$ for i in `gitlog.id2 recovery25` ; do git cherry-pick $i ;don
```

4. gitlog.grep

gitlog.grep greps for a string within that collection of patches. For example, if I find a bug and want to fix the patch that has a reference to function **inode_go_sync**, I simply do:

```
$ gitlog.grep inode_go_sync
-----------------------[ recovery25 - 50 patches ]-------------
(snip)
11 340d27a33895 Bob Peterson     gfs2: drain the ail2 list afte
```

```
10 9b3c4e6efb10 Bob Peterson     gfs2: clean up iopen glock mes
 9 d2e8c22be39b Bob Peterson     gfs2: Do proper error checking
152:-static void inode_go_sync(struct gfs2_glock *gl)
153:+static int inode_go_sync(struct gfs2_glock *gl)
163:@@ -296,6 +302,7 @@ static void inode_go_sync(struct gfs2_g
 8 9563e31f8bfd Christoph Hellwig gfs2: use page_offset in gfs2
 7 ebac7a38036c Christoph Hellwig gfs2: don't use buffer_heads
 6 f703a3c27874 Andreas Gruenbacher gfs2: Improve mmap write vs
 5 a3e86d2ef30e Andreas Gruenbacher gfs2: Multi-block allocatio
 4 da3c604755b0 Andreas Gruenbacher gfs2: Fix end-of-file handl
 3 4525c2f5b46f Bob Peterson     Rafael Aquini's slab instrumen
 2 a06a5b7dea02 Bob Peterson     GFS2: Add go_get_holdtime to g
 ^ 8ba93c796d5c Bob Peterson     gfs2: introduce new function r
```

So, now I know that patch **HEAD~9** is the one that needs fixing. I use **git rebase -i HEAD~10** to edit patch 9, **git commit -a --amend**, then **git rebase --continue** to make the necessary adjustments.

5. gitbranchcmp3

gitbranchcmp3 lets me compare my current branch to another branch, so I can compare older versions of patches to my newer versions and quickly see what's changed and what hasn't. It generates a compare script (that uses the KDE tool Kompare, which works on GNOME3, as well) to compare the patches that aren't quite the same. If there are no differences other than line numbers, it prints **[SAME]**. If there are only comment differences, it prints **[same]** (in lower case). For example:

```
$ gitbranchcmp3 recovery24
Branch recovery24 has 47 patches
Branch recovery25 has 50 patches

(snip)
38 87eb6901607a 340d27a33895 [same] gfs2: drain the ail2 list a
39 90fefb577a26 9b3c4e6efb10 [same] gfs2: clean up iopen glock
40 ba3ae06b8b0e d2e8c22be39b [same] gfs2: Do proper error check
```

```
41 2ab662294329 9563e31f8bfd [SAME] gfs2: use page_offset in gf
42 0adc6d817b7a ebac7a38036c [SAME] gfs2: don't use buffer_head
43 55ef1f8d0be8 f703a3c27874 [SAME] gfs2: Improve mmap write vs
44 de57c2f72570 a3e86d2ef30e [SAME] gfs2: Multi-block allocatio
45 7c5305fbd68a da3c604755b0 [SAME] gfs2: Fix end-of-file handl
46 162524005151 4525c2f5b46f [SAME] Rafael Aquini's slab instru
47              a06a5b7dea02 [    ] GFS2: Add go_get_holdtime t
48              8ba93c796d5c [    ] gfs2: introduce new functio
49              e8b5ff851bb9 [    ] gfs2: Allow rgrps to have a

Missing from recovery25:
The missing:
Compare script generated at: /tmp/compare_mismatches.sh
```

6. gitlog.find

Finally, I have **gitlog.find**, a script to help me identify where the upstream versions of my patches are and each patch's current status. It does this by matching the patch description. It also generates a compare script (again, using Kompare) to compare the current patch to the upstream counterpart:

```
$ gitlog.find
-----------------------[ recovery25 - 50 patches ]-------------
(snip)
11 340d27a33895 Bob Peterson     gfs2: drain the ail2 list afte
lo 5bcb9be74b2a Bob Peterson     gfs2: drain the ail2 list afte
10 9b3c4e6efb10 Bob Peterson     gfs2: clean up iopen glock mes
fn 2c47c1be51fb Bob Peterson     gfs2: clean up iopen glock mes
 9 d2e8c22be39b Bob Peterson     gfs2: Do proper error checking
lo feb7ea639472 Bob Peterson     gfs2: Do proper error checking
 8 9563e31f8bfd Christoph Hellwig gfs2: use page_offset in gfs2
ms f3915f83e84c Christoph Hellwig gfs2: use page_offset in gfs2
 7 ebac7a38036c Christoph Hellwig gfs2: don't use buffer_heads
ms 35af80aef99b Christoph Hellwig gfs2: don't use buffer_heads
 6 f703a3c27874 Andreas Gruenbacher gfs2: Improve mmap write vs
fn 39c3a948ecf6 Andreas Gruenbacher gfs2: Improve mmap write vs
```

```
 5 a3e86d2ef30e Andreas Gruenbacher gfs2: Multi-block allocatio
fn f53056c43063 Andreas Gruenbacher gfs2: Multi-block allocatio
 4 da3c604755b0 Andreas Gruenbacher gfs2: Fix end-of-file handl
fn 184b4e60853d Andreas Gruenbacher gfs2: Fix end-of-file handl
 3 4525c2f5b46f Bob Peterson        Rafael Aquini's slab instrumen
   Not found upstream
 2 a06a5b7dea02 Bob Peterson        GFS2: Add go_get_holdtime to g
   Not found upstream
 ^ 8ba93c796d5c Bob Peterson        gfs2: introduce new function r
   Not found upstream
 H e8b5ff851bb9 Bob Peterson        gfs2: Allow rgrps to have a mi
   Not found upstream
```

The patches are shown on two lines, the first of which is your current patch, followed by the corresponding upstream patch, and a 2-character abbreviation to indicate its upstream status:

- **lo** means the patch is in the local upstream Git repo only (i.e., not pushed upstream yet).
- **ms** means the patch is in Linus Torvald's master branch.
- **fn** means the patch is pushed to my "for-next" development branch, intended for the next upstream merge window.

Some of my scripts make assumptions based on how I normally work with Git. For example, when searching for upstream patches, it uses my well-known Git tree's location. So, you will need to adjust or improve them to suit your conditions. The **gitlog.find** script is designed to locate GFS2 and DLM patches only, so unless you're a GFS2 developer, you will want to customize it to the components that interest you.

Source code

Here is the source for these scripts.

1. gitlog

```
#!/bin/bash
branch=$1

if test "x$branch" = x; then
    branch=`git branch -a | grep "*" | cut -d ' ' -f2`
fi

patches=0
tracking=`git rev-parse --abbrev-ref --symbolic-full-name

LIST=`git log --reverse --abbrev-commit --pretty=oneline $t
for i in $LIST; do patches=$(echo $patches + 1 | bc);done

if [[ $branch =~ .*for-next.* ]]
then
    start=HEAD
#   start=origin/for-next
else
    start=origin/master
fi

tracking=`git rev-parse --abbrev-ref --symbolic-full-name

/usr/bin/echo "-----------------------[" $branch "]-----------------
patches=$(echo $patches - 1 | bc);
for i in $LIST; do
    if [ $patches -eq 1 ]; then
        cnt=" ^"
    elif [ $patches -eq 0 ]; then
        cnt=" H"
    else
        if [ $patches -lt 10 ]; then
            cnt=" $patches"
        else
            cnt="$patches"
        fi
    fi
    /usr/bin/git show --abbrev-commit -s --pretty=format:"
    patches=$(echo $patches - 1 | bc)
done
#git log --reverse --abbrev-commit --pretty=format:"%h %<|(32)%an %s" $t
#git log --reverse --abbrev-commit --pretty=format:"%h %<|(32)%an %s" ^o
```

2. gitlog.id

```
#!/bin/bash
branch=$1

if test "x$branch" = x; then
    branch=`git branch -a | grep "*" | cut -d ' ' -f2`
fi

tracking=`git rev-parse --abbrev-ref --symbolic-full-name

/usr/bin/echo "----------------------[" $branch "]----------------
git log --reverse --abbrev-commit --pretty=oneline $tracking
```

3. gitlog.id2

```
#!/bin/bash
branch=$1

if test "x$branch" = x; then
    branch=`git branch -a | grep "*" | cut -d ' ' -f2`
fi

tracking=`git rev-parse --abbrev-ref --symbolic-full-name
git log --reverse --abbrev-commit --pretty=oneline $tracking
```

4. gitlog.grep

```
#!/bin/bash
param1=$1
param2=$2

if test "x$param2" = x; then
    branch=`git branch -a | grep "*" | cut -d ' ' -f2`
    string=$param1
else
    branch=$param1
    string=$param2
fi

patches=0
tracking=`git rev-parse --abbrev-ref --symbolic-full-name

LIST=`git log --reverse --abbrev-commit --pretty=oneline $t
for i in $LIST; do patches=$(echo $patches + 1 | bc);done
```

```
/usr/bin/echo "----------------------[" $branch "-" $patches "patche
patches=$(echo $patches - 1 | bc);
for i in $LIST; do
    if [ $patches -eq 1 ]; then
        cnt=" ^"
    elif [ $patches -eq 0 ]; then
        cnt=" H"
    else
        if [ $patches -lt 10 ]; then
            cnt=" $patches"
        else
            cnt="$patches"
        fi
    fi
    /usr/bin/git show --abbrev-commit -s --pretty=format:"
    /usr/bin/git show --pretty=email --patch-with-stat $i
    patches=$(echo $patches - 1 | bc)
done
```

5. gitbranchcmp3

```
#!/bin/bash
#
# gitbranchcmp3 <old branch> [<new_branch>]
#
oldbranch=$1
newbranch=$2
script=/tmp/compare_mismatches.sh

/usr/bin/rm -f $script
echo "#!/bin/bash" > $script
/usr/bin/chmod 755 $script
echo "# Generated by gitbranchcmp3.sh" >> $script
echo "# Run this script to compare the mismatched patches" >> $script
echo " " >> $script
echo "function compare_them()" >> $script
echo "{"  >> $script
echo "    git show --pretty=email --patch-with-stat \$1 > /tmp/gronk1" 
echo "    git show --pretty=email --patch-with-stat \$2 > /tmp/gronk2" 
echo "    kompare /tmp/gronk1 /tmp/gronk2" >> $script
echo "}" >> $script
echo " " >> $script

if test "x$newbranch" = x; then
    newbranch=`git branch -a | grep "*" | cut -d ' ' -f2`
fi
```

```
tracking=`git rev-parse --abbrev-ref --symbolic-full-name

declare -a oldsha1s=(`git log --reverse --abbrev-commit --pr
declare -a newsha1s=(`git log --reverse --abbrev-commit --pr

#echo "old: " $oldsha1s
oldcount=${#oldsha1s[@]}
echo "Branch $oldbranch has $oldcount patches"
oldcount=$(echo $oldcount - 1 | bc)
#for o in `seq 0 ${#oldsha1s[@]}`; do
#    echo -n ${oldsha1s[$o]} " "
#    desc=`git show $i | head -5 | tail -1|cut -b5-`
#done

#echo "new: " $newsha1s
newcount=${#newsha1s[@]}
echo "Branch $newbranch has $newcount patches"
newcount=$(echo $newcount - 1 | bc)
#for o in `seq 0 ${#newsha1s[@]}`; do
#    echo -n ${newsha1s[$o]} " "
#    desc=`git show $i | head -5 | tail -1|cut -b5-`
#done
echo

for new in `seq 0 $newcount`; do
    newsha=${newsha1s[$new]}
    newdesc=`git show $newsha | head -5 | tail -1|cut -b5-`
    oldsha="          "
    same="[    ]"
    for old in `seq 0 $oldcount`; do
        if test "${oldsha1s[$old]}" = "match"; then
            continue;
        fi
        olddesc=`git show ${oldsha1s[$old]} | head -5 | tail
        if test "$olddesc" = "$newdesc" ; then
            oldsha=${oldsha1s[$old]}
            #echo $oldsha
            git show $oldsha |tail -n +2 |grep -v "index.*\.\
            git show $newsha |tail -n +2 |grep -v "index.*\.\
            diff /tmp/gronk1 /tmp/gronk2 &> /dev/null
            if [ $? -eq 0 ] ;then
# No differences
                same="[SAME]"
                oldsha1s[$old]="match"
                break
            fi
            git show $oldsha |sed -n '/diff/,$p' |grep -v "ind
            git show $newsha |sed -n '/diff/,$p' |grep -v "ind
            diff /tmp/gronk1 /tmp/gronk2 &> /dev/null
```

```
               if [ $? -eq 0 ] ;then
# Differences in comments only
                    same="[same]"
                    oldsha1s[$old]="match"
                    break
               fi
               oldsha1s[$old]="match"
               echo "compare_them $oldsha $newsha" >> $script
          fi
     done
     echo "$new $oldsha $newsha $same $newdesc"
done

echo
echo "Missing from $newbranch:"
the_missing=""
# Now run through the olds we haven't matched up
for old in `seq 0 $oldcount`; do
     if test ${oldsha1s[$old]} != "match"; then
          olddesc=`git show ${oldsha1s[$old]} | head -5 | tail
          echo "${oldsha1s[$old]} $olddesc"
          the_missing=`echo "$the_missing ${oldsha1s[$old]}"`
     fi
done

echo "The missing: " $the_missing
echo "Compare script generated at: $script"
#git log --reverse --abbrev-commit --pretty=oneline $tracking..$branch |
```

6. gitlog.find

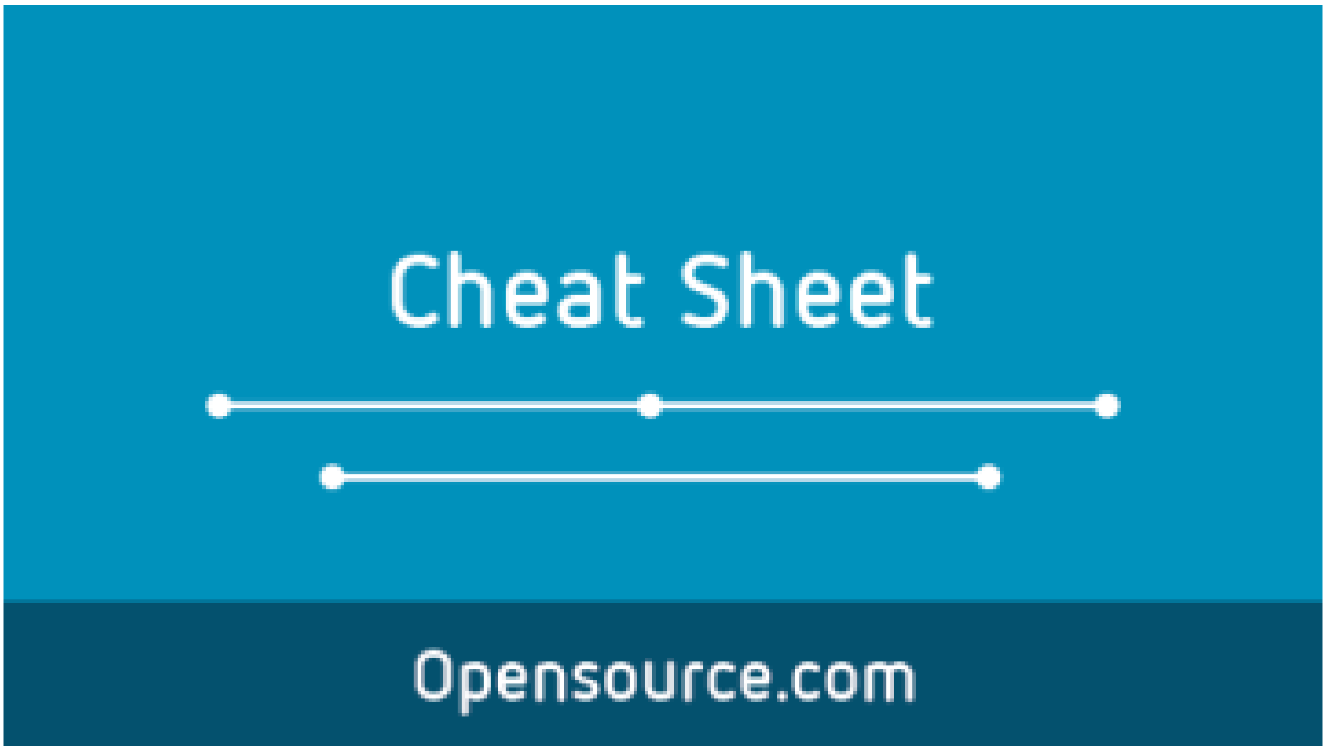
```
#!/bin/bash
#
# Find the upstream equivalent patch
#
# gitlog.find
#
cwd=$PWD
param1=$1
ubranch=$2
patches=0
script=/tmp/compare_upstream.sh
echo "#!/bin/bash" > $script
/usr/bin/chmod 755 $script
echo "# Generated by gitbranchcmp3.sh" >> $script
echo "# Run this script to compare the mismatched patches" >> $script
echo " " >> $script
echo "function compare_them()" >> $script
echo "{"  >> $script
echo "    cwd=$PWD" >> $script
echo "    git show --pretty=email --patch-with-stat \$2 > /tmp/gronk2" 
echo "    cd ~/linux.git/fs/gfs2" >> $script
echo "    git show --pretty=email --patch-with-stat \$1 > /tmp/gronk1" 
echo "    cd $cwd" >> $script
echo "    kompare /tmp/gronk1 /tmp/gronk2" >> $script
echo "}" >> $script
echo " " >> $script

#echo "Gathering upstream patch info. Please wait."
branch=`git branch -a | grep "*" | cut -d ' ' -f2`
tracking=`git rev-parse --abbrev-ref --symbolic-full-name

cd ~/linux.git
if test "X${ubranch}" = "X"; then
    ubranch=`git branch -a | grep "*" | cut -d ' ' -f2`
fi
utracking=`git rev-parse --abbrev-ref --symbolic-full-name
#
# gather a list of gfs2 patches from master just in case we can't find i
#
#git log --abbrev-commit --pretty=format:"   %h %<|(32)%an %s" master |g
git log --reverse --abbrev-commit --pretty=format:"ms %h %<|
# ms = in Linus's master
git log --reverse --abbrev-commit --pretty=format:"ms %h %<|

cd $cwd
LIST=`git log --reverse --abbrev-commit --pretty=oneline $
for i in $LIST; do patches=$(echo $patches + 1 | bc);done
/usr/bin/echo "-----------------------[" $branch "-" $patches "patche
patches=$(echo $patches - 1 | bc);
for i in $LIST; do
    if [ $patches -eq 1 ]; then
        cnt=" ^"
    elif [ $patches -eq 0 ]; then
        cnt=" H"
    else
        if [ $patches -lt 10 ]; then
            cnt=" $patches"
        else
            cnt="$patches"
        fi
    fi
    /usr/bin/git show --abbrev-commit -s --pretty=format:"
    desc=`/usr/bin/git show --abbrev-commit -s --pretty=fo
    cd ~/linux.git
    cmp=1
    up_eq=`git log --reverse --abbrev-commit --pretty=form
# lo = in local for-next
    if test "X$up_eq" = "X"; then
        up_eq=`git log --reverse --abbrev-commit --pretty=
# fn = in for-next for next merge window
        if test "X$up_eq" = "X"; then
            up_eq=`grep "$desc" /tmp/gronk.gfs2`
            if test "X$up_eq" = "X"; then
                up_eq=`grep "$desc" /tmp/gronk.dlm`
                if test "X$up_eq" = "X"; then
                    up_eq="   Not found upstream"
                    cmp=0
                fi
            fi
        fi
    fi
    echo "$up_eq"
    if [ $cmp -eq 1 ] ; then
        UP_SHA1=`echo $up_eq|cut -d' ' -f2`
        echo "compare_them $UP_SHA1 $i" >> $script
    fi
    cd $cwd
    patches=$(echo $patches - 1 | bc)
done
echo "Compare script generated: $script"
```

What to read next



Bash cheat sheet: Key combos and special syntax

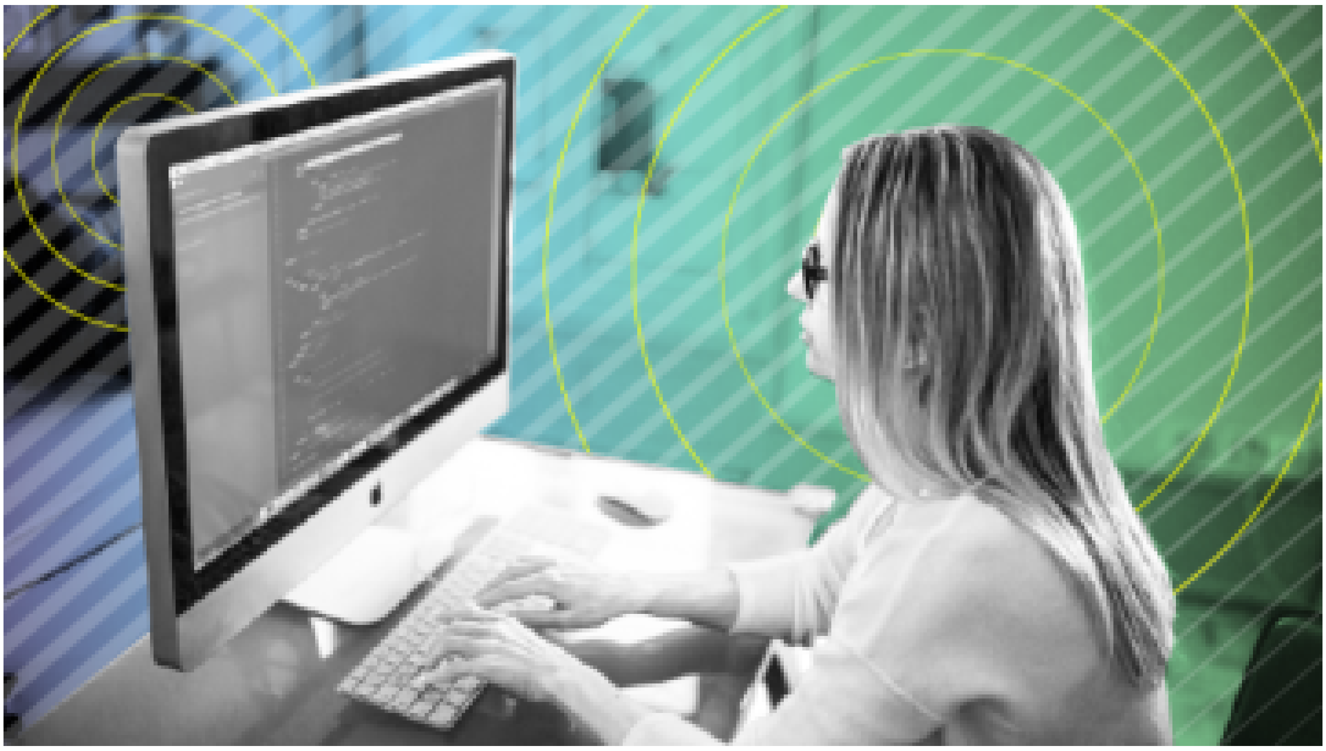
Download our new cheat sheet for Bash commands and shortcuts you need to talk to your computer.

Seth Kenlon (Team, Red Hat)

10 resources to boost your Git skills

Wrap up the year with a review of the best in Git. Here are the top 10 articles about surprising ways you can use Git in the new year.

Joshua Allen Holm (Alumni)



Creating a Bash script template

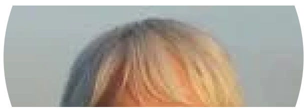
In the second article in this series, create a fairly simple template that you can use as a starting point for other Bash programs, then test it.

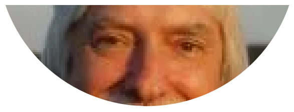
David Both (Correspondent)

Tags: SCRIPTING BASH GIT

Bob Peterson

Bob Peterson is a Linux kernel developer working for Red Hat on the GFS2 file system.





More about me

2 Comments

These comments are closed.

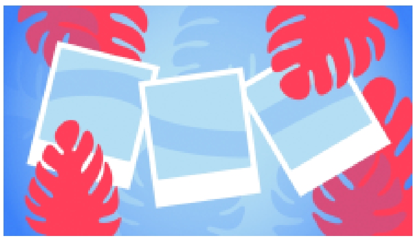
MaxOd | January 30, 2020
No readers like this yet.

`gitbranchcmp3`: you may find built-in `git range-diff` useful

John Me | February 4, 2020
No readers like this yet.

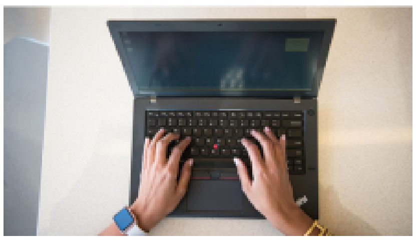
It is a pity that the scripts have not been checked with 'shellcheck' -> there are several errors (on Debian/Ubuntu there is /bin/echo, no /usr/bin/echo like on RedHat), inconsistencies (git and /usr/bin/git),...

Related Content



Automate image processing with this Bash script

Use this Python script to simulate



Using Bash traps in your scripts

Babbage's
Difference Engine