Advanced Git Commands You Will Actually Use

Tom Hacohen
Samsung Open Source Group
tom.hacohen@samsung.com
@TomHacohen

stosb.com/talks
Quick Setup Before We Begin

- In case you do not have these set already:
  
  ```
  $ git config --global user.name "Tom Hacohen"
  $ git config --global user.email "tom.hacohen@samsung.com"
  $ git config --global color.ui true
  $ export EDITOR="vim"  # Optional
  $ git config --global diff.tool "vimdiff"  # Extra optional
  ```

  ```
  $ git config --global alias.unstage 'reset HEAD --'
  ```

- Enable Git command autocompletion (system dependent)
- Set up aliases (here for reference)
A Few Useful Properties of Git

Snapshots, Not Differences

- Differences

- Snapshots
A Few Useful Properties of Git

Branches and Tags are References
A Few Useful Properties of Git

Nearly Every Operation Is Local

- Operations are fast
- You can work offline
- You can be sloppy (and fix later)
A Few Useful Properties of Git

Git Generally Only Adds Data

- Easy to see older (even temporary) versions
- Almost everything is reversible
- Safe to experiment with
A Few Useful Properties of Git

Just a Few More…

• Has a staging area
• History is mutable (easy to abuse)
• Everything is checksummed
$ git log --oneline --graph
* 7b3fa8a Migrate elementary to the new Eo4 syntax
\* 02e87e8 Fix warnings following migration to Eo4.
\* 7bd1d48 Map: Correct broken migration.
\* e74ec8c Automatic migration to Eo4.
\* 9274efb Remove redundant defines.
/
* 36669f1 Scaling test: reorder instructions to set the correct scale
* b9c912f radio: inherit from elm check
Reading the Log

Listing a Change Summary

$ git log --stat
commit a31f399857ecf9409e6aa6fb8effe9477ee47fe2
Author: Tom Hacohen <tom@stosb.com>
Date: Wed Jun 22 16:52:05 2016 +0100

    Edje object: Add API for replacing the internal text object

    src/lib/edje/edje_load.c    | 34 ++++++++++++++++++++++++++++++++++++++
    src/lib/edje/edje_object.eo| 17 +++++++++++++++++++++++++++++++++++
    src/lib/edje/edje_private.h| 1 +
    src/lib/edje/edje_util.c   | 32 ++++++++++++++++++++++++++++++++++++++++

4 files changed, 60 insertions(+), 24 deletions(-)
Reading the Log

Seeing Actual Code Changes

$ git log -p
commit c66d478ebb2c1a0aff52571467d7a6b87a533382
Author: Ji-Youn Park <jy0703.park@samsung.com>
Date:   Thu Mar 24 17:54:05 2016 +0830

    Elm_image: remove Elm_Image_Orient.

diff --git a/src/lib/elm_image.c b/src/lib/elm_image.c
*********** Snip Snip ***********
}

  EOLIAN static void
- _elm_image_orient_set(Eo *obj, Elm_Image_Data *sd, Elm_Image_Orient orient)
+ elm_image_efl_image_orientation_set(Eo *obj, Elm_Image_Data *sd, Efl_Gfx_Orientation orient)
{
    if (sd->edje) return;
Reading the Log

Limiting Commits

# Commits changing file/function in file
$ git log main.c
$ git log -L :list_find:main.c

# Commits containing string
$ git log --grep FOOBAR # Messages containing string
$ git log -S FOOBAR # Lines containing string

# Commits in HEAD, not in master
$ git log master..

# Commits that are in either "foo" or "bar" (not both)
$ git log foo...bar

# Only show the first parent ("main" history)
$ git log --first-parent
Reading the Log

Finding Out What You Have Been Up To

# The total number of commits by an author
$ git shortlog -nse --author=tom.hacohen@samsung.com

# The total number of commits by an author in the last year
$ git shortlog -nse --author=tom --since="1 year ago"

# A list of commits by an author in the last year
$ git log --author=tom --since="1 year ago"
Reading the Log

Getting a Version Description

# Get a standard version description (requires at least one tag)
$ git describe --long
v1.17.0-236-gc66d478

# Get the SVN-like monotonic revision number
$ git rev-list --count HEAD
12514
Inspecting Commits and States

Seeing Into the Past

# Showing the changes in a commit
$ git show 80f14e8fea0057ee950f0778dd51b096ca9850a4
$ git show my_branch # Can be branch, tag or whatever.

# Showing a file from a different state
$ git show v1.7.0:main.c

# Switching working directory to a different reference
$ git checkout c9b306777 # Or any other reference
Branches

Viewing

# All the branches (including remote)
$ git branch -a

# Use "git fetch -p" to clean up stale remote branches

# All branches that are fully contained in HEAD
$ git branch -a --merged

# All branches that are not full contained in HEAD
$ git branch -a --no-merged
Branches

Manipulating

# Rebase branch over the upstream version
$ git pull --rebase # Can be set in config

# Rebase branch over a specific branch
$ git rebase origin/master

# Merge a branch and always create a merge commit
$ git merge --no-ff

# Rebase and keep the branch structure
$ git pull --rebase=preserve
$ git rebase --preserve-merges origin/master

# Applying a commit from a different branch
$ git cherry-pick 80f122437d
Making Changes

Inspecting Workspace State

# A more condensed status
$ git status -s

# Changes compare to upsteram
$ git diff origin/master

# Seeing the diff of the staging area
$ git diff --cached

# Ignore whitespace changes in diff
$ git diff -w
Making Changes

Adding Files to the Staging Area

# Adding parts of a file
$ git add -p file # File can also be a dir, or omitted

# Adding all of the changed files in a directory
$ git add -u src/
Making Changes

Using the Stash

# Stashing all of the changes
$ git stash

# Stashing some of the changes
$ git stash -p

# Applying back the stash
$ git stash apply

# Stash has many more features I do not use
$ git stash --help
Rewriting History

Un-staging Files

```bash
$ git status -s
M README

$ git reset README
# git reset # for all the files
$ git status -s
M README

$ git checkout README
# git checkout -f # for all the files
$ git status -s
# Nothing
```
Rewriting History

Editing the Most Recent Commits

# Remove the most recent commits and their changes
$ git reset --hard HEAD^
$ git reset --hard HEAD~3 # Or any other pointer (for a range)
$ git reset --hard origin/master # Reset the state to upstream

# Keep the changes uncommitted
$ git reset HEAD^
$ git reset c9b306777 # Or any other pointer

# Merging index into the most recent commit
$ git add NEWS
$ git commit --amend # Also lets you edit the commit message
# Add -v to git commit to also see the diff

# Edit the author
$ git commit --author "007 <jb@mi6.gov.uk>" --amend
Rewriting History

The Most Useful Command in The World

$ git rebase -i HEAD~5
pick 7b07b03 track/manage size hints for zoomap child objects
pick 71a85b7 update winlist ui when using directional selection
pick bbd4d2f force changed when adding keyboards
pick a424542 disable emotion_shutdown during shutdown procedure

# Commands:
# p, pick = use commit
# r, reword = use commit, but edit the commit message
# e, edit = use commit, but stop for amending
# s, squash = use commit, but meld into previous commit
# f, fixup = like "squash", but discard this commit's log message
# x, exec = run command (the rest of the line) using shell
# d, drop = remove commit
#
# These lines can be re-ordered; executed from top to bottom.
# If you remove a line here THAT COMMIT WILL BE LOST.
Rewriting History

Recovering Lost Commits

# Jump to a hash state if you know it
$ git checkout c9b306777

# Find unreferenced (missing) commits
$ git reflog
cebf78d HEAD@{0}: rebase -i (finish): returning to refs/heads/master
cebf78d HEAD@{1}: rebase -i (start): checkout HEAD^^^^
c6e355f HEAD@{2}: rebase finished: returning to refs/heads/master
c6e355f HEAD@{3}: pull --rebase: Elementary test entry: Create an editable test object.
83b0592 HEAD@{4}: commit: Elementary test entry: Create an editable test object.
2fd8861 HEAD@{5}: commit (amend): Ui text: Add an editable variant (tiny wrapper).
Rewriting History

Removing Parts of a Commit

# Commit c42bc3a535 (can be anywhere in history)
$ git revert -n c42bc3a535
$ git reset # Remove everything from staging

# Add back the wanted changes
$ git add NEWS # All of this file
$ git add -p # Some parts of the rest

# Merge the commit into the original commit
# Either amend if it is the HEAD
$ git commit --amend
$ git checkout -f # Remove the rest of the changes
# Or fixup if anywhere else
$ git commit -m "Temp"
$ git checkout -f # Remove the rest of the changes
$ git rebase -i c42bc3a535^ # Mind the ^ (caret)
Delivering Changes

Getting Your Commits Out There

# Change the url of the repository
$ git remote set-url origin ssh://git@newserver.com/repo.git

# Adding a new remote
$ git remote add new ssh://git@alt.newserver.com/repo.git

# Using the new remote
$ git fetch new
$ git rebase new/master
$ git push new master

# Generate patch files for a series of commits
$ git format-patch HEAD~5  # Or any other reference
Investigating Bugs
Finding Who Added a Line and Why

# Check who changed the file
$ git blame eo.c # Add "-w" to ignore white-space

06f65ab2 eo.c (Tom Hacohen 2016-05-19 11:33:17 +0100
321)       vtable = &klass->vtable;
fc880379 eo.c (Tom Hacohen 2015-11-09 11:45:04 +0000
322)       inputklass = main_klass = klass;
7be0748b eo.c (Jérémy Zurcher 2013-07-30 15:02:35 +0200
323)
c2b4137f eo.c (Carsten Haitzler 2015-10-24 12:23:53 +0900
324) if (!cache->op)

# At an earlier revision
$ git blame fc880379^ -- eo.c
Investigating Bugs
Finding When a Bug Was Introduced

$ git bisect start
# To limit bisect to a directory: "git bisect start -- src/"
$ git bisect bad COMMIT
$ git bisect good COMMIT
Bisecting: 417 revisions left to test after this (roughly 9 steps)
[7352bcff98fc65a08edcd505b872403af8d821a7] edje_external: fix external icon handling
$ git bisect good
Bisecting: 208 revisions left to test after this (roughly 8 steps)
[9f5d27972252d67fe92ca44a1c610da4ed531b86] Evas events: Implement support for hold event

# ... SNIP ...

a31f399857ecf9409e6aa6fb8effe9477ee47fe2 is the first bad commit
Investigating Bugs

Automatic Bisect

```
$ cat ~/test.sh
#!/bin/sh
make || exit 125          # this skips broken builds
~/check_issue.sh          # does the test case pass?

$ git bisect start HEAD HEAD~10 --   # Last 10 commits
$ git bisect run ~/test.sh
$ git bisect reset        # quit the bisect session

a31f399857ecf9409e6aa6fb8effe9477ee47fe2 is the first bad commit
```
A Useful Tool - tig

Main View
A Useful Tool - tig

Blame
Name Dropping

A Few Commands You Should Read More About

- git send-email
- git checkout -b
- git-svn, git-hg...
- gitg
- gitk
HIJACKING THIS TALK

EVERYBODY OUT.

THIS IS MY THREAD NOW.
A Better Workflow

“I messed up Git so bad it turned into Guitar Hero”
A Better Workflow

Non-Linear vs. Linear
A Better Workflows

Maintaining a Linear History

# Maybe alias some of these
# Sync with upstream
$ git pull --rebase=preserve

# Merging a branch into master (working on branch feature)
$ git rebase --preserve-merges master
$ git checkout master
$ git merge --no-ff feature # Remove --no-ff for no merge commit
Commit Messages

Style Guideline

- Wrap message around 76 characters
- First line: brief, one-liner
- Second line should be left empty
- The body of the commit should be descriptive and verbose
- Use the imperative form for consistency with git merge and revert ("Fix bug" not "Fixed bug")
- Bonus: annotate with @fix, @feature, #NNN and etc.
Commit Messages

Example

Eo refcount: Split the refcount to private and public (user).

This commit changes the way refcount is dealt with internally. Before this commit, there was one refcount shared between Eo internals and users. Now there is a refcount for eo operations (like for example, function calls) and one for user refcount (eo_ref).

An example bug that this protects against (which is seemingly rather common) is:

```c
some_eo_func(obj);
```

// Inside the implementation of that func:
pd->a = 1; // The object's private data
eo_unref(obj); // To delete the object
eo_unref(obj); // A big one extra_unref
pd->a = 2; // Segfault, this data has already been freed

T3428
@feature
Questions?

stosb.com/talks

Tom Hacohen
Samsung Open Source Group
tom.hacohen@samsung.com
@TomHacohen
Attribution

- The Git Book
- Workflows Image #1
- Workflows Image #2
- Hijacking this talk