Git-ing out of your Git messes

Katie Sylor-Miller
@ksylor && @ohshitgit
Etsy is a global marketplace where people around the world connect, both online and offline, to make, sell and buy unique goods.
I made a website to explain how to get yourself out of your git messes in plain English

ohshitgit.com
Don’t git into a mess in the first place
Agenda

- Fundamental Concepts
- Workflow in-depth
- Viewing and changing history
- Avoiding (and fixing) messes
Fundamental concepts

Commits
Branches
HEAD
History
Fundamental concepts

Commits
Branches
HEAD
History
What’s in a commit

Each commit contains a few pieces of information:

• A snapshot of the entire repo
• Who made this change
• When this change was made
• A message describing the commit
• A pointer to the previous commit
What’s in a commit

Each commit contains a few pieces of information:

- A snapshot of the entire repo
- Who made this change
- When this change was made
- A message describing the commit
- A pointer to the previous commit
Commit hash abbreviations

- 912bde5
- a4df41a
- aec5611
- 6dab6a7
- 1a35312
Fundamental concepts

Commits
Branches
HEAD
History
Mental model: branches are a linked list
Mental model: branches are a linked list

- A snapshot of the entire repo
- Who made this change
- When this change was made
- A message describing the commit
- A pointer to the previous commit
Mental model: branches are a linked list

- A snapshot of the entire repo
- Who made this change
- When this change was made
- A message describing the commit
- A pointer to the previous commit
Branches are a linked list reference

1a35312 → 6dab6a7 → aec5611 → a4df41a → 912bde5

master
Branches are a linked list reference text file

```
.git/refs/heads/master
912bde5d4e5e962269ddff87da83cc5ce55e75d0
```
Create a new branch

ksylormiller:~/ohshitgit (master)$ git branch new-branch
New branches are a new reference
New branches are a new text file

```
.git/refs/heads/new-branch
912bde5d4e5e962269d5f87da83cc5ce55e75d0
```

```
.git/refs/heads/master
912bde5d4e5e962269d5f87da83cc5ce55e75d0
```

Diagram showing the relationship between branches and commit hashes.
Fundamental concepts

Commits
Branches
HEAD
History
HEAD points to your currently checked-out branch
HEAD is a reference to a reference

```plaintext
.git/refs/HEAD
ref: refs/heads/master
```
WTF is the HEAD

- The HEAD decides what branch/commit will be the target of your actions
- There is only one HEAD per repository
- The HEAD is unique to your local install of git
- The history of the HEAD is recorded, which gives you a way to undo your work (more on this later)
Change branches

ksylormiller:~/ohshitgit (master)$ git checkout new-branch
HEAD points to your currently checked-out branch

.git/refs/HEAD
ref: refs/heads/master

B

C

D

E

6dab6a7

aec5611

a4df41a

912bde5

new-branch

HEAD

master
Add a new commit to our branch

ksylormiller:~/ohshitgit (master)$ git commit -m “Your message here”
A new commit becomes the new HEAD

```
B 6dab6a7
C aec5611
D a4df41a
E 912bde5
F 1668d2f
```

```
HEAD
new-branch
master
```
Fundamental concepts

Commits
Branches
HEAD
History
History is a linked list
History is a linked list tree
History is a linked list tree directed acyclic graph
Environments
A simple workflow
A complicated workflow
Environments

A simple workflow

A complicated workflow
<table>
<thead>
<tr>
<th>Github</th>
<th>Your machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE/ORIGIN</td>
<td>LOCAL</td>
</tr>
<tr>
<td>Central server where</td>
<td>STAGING/INDEX</td>
</tr>
<tr>
<td>shared git repositories</td>
<td>WORKSPACE</td>
</tr>
<tr>
<td>are stored</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Remotes typically are</td>
<td></td>
</tr>
<tr>
<td>“bare” - you can’t</td>
<td></td>
</tr>
<tr>
<td>directly modify them</td>
<td></td>
</tr>
<tr>
<td>Github</td>
<td>Your machine</td>
</tr>
<tr>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>REMOTE/ORIGIN</td>
<td>LOCAL</td>
</tr>
</tbody>
</table>

- Your local copy of the remote git repository
- Lives in a .git folder
- Contains the entire history of a remote repo plus local changes
Snapshot of changes to the current branch that you want to commit

--

Is a copy of all of the files in the repo, not just the changed files
<table>
<thead>
<tr>
<th>Github</th>
<th>Your machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>REMOTE/ORIGIN</td>
<td>LOCAL STAGING/INDEX WORKSPACE</td>
</tr>
<tr>
<td></td>
<td>Where changes to files are made</td>
</tr>
<tr>
<td></td>
<td>--</td>
</tr>
<tr>
<td></td>
<td>Analogous to the physical directory where files are stored on disk</td>
</tr>
</tbody>
</table>
Workflows in depth

Environments

A simple workflow

A complicated workflow
REMOTE/ORIGIN

GitHub

REMOTE/ORIGIN

LOCAL

Your machine

STAGING/INDEX

baz.php

foo.php

bar.php

您的机器

STAGING/INDEX

baz.php

foo.php

bar.php

WORKSPACE

bar.php

baz.php
git status

is your BFF

by Etsy seller SmittenKittenKendall
git status is your BFF

ksylormiller:~/ohshitgit (master)$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

      modified: bar.php

no changes added to commit (use "git add" and/or "git commit --a")
git add

ksylormiller:~/ohshitgit (master)$ git add bar.php
git commit

ksylormiller:~/ohshitgit (master)$ git commit -m "Meaningful commit message"
[master 4875652] Meaningful commit message
 1 file changed, 1 insertion(+)
git push

ksylormiller:~/ohshitgit (master)$ git push
Counting objects: 4, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 328 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/ksylor/fluent-example.git
  4875652..97e8858  master -> master
But... it’s usually not that simple
Workflows in depth

Environments

A simple workflow

A complicated workflow
Github
REMOTE/ORIGIN

LOCAL
MASTER

STAGING/INDEX
MASTER

WORKSPACE
ksylormiller:~/ohshitgit (master)$ git fetch
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/ksylor/fluent-example
  97e8858..8a97966  master    -> origin/master
git merge

ksylormiller:~/ohshitgit (master)$ git merge origin/master
oh god we're in vim

Merge remote-tracking branch 'origin/master'

# Please enter a commit message to explain why this merge is necessary,
# especially if it merges an updated upstream into a topic branch.
#
# Lines starting with '#' will be ignored, and an empty message aborts
# the commit.
~
~
~
~
~
"~/ohshitgit/.git/MERGE_MSG" 7L, 293C
ksylormiller:~/ohshitgit (master)$ git merge origin/master
Updating 97e8858..8a97966
Fast-forward
  README.md | 2 ++
1 file changed, 2 insertions(+)
create mode 100644 README.md

git merge complete
git pull == git fetch && git merge

ksylormiller:~/ohshitgit (master)$ git pull
Merge made by the 'recursive' strategy.
  README.md  |  2 ++
1 file changed, 2 insertions(+) 
create mode 100644 README.md
ksylormiller:~/ohshitgit (master)$ git push
Counting objects: 8, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 537 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To https://github.com/ksylor/fluent-example.git
  8a97966..91f6790  master --> master
Rebase all the things!
git rebase

ksylormiller:~/ohshitgit (master)$ git rebase origin/master
First, rewinding head to replay your work on top of it...
Applying: Meaningful commit message
git pull --rebase == git fetch && git rebase

ksylormiller:~/ohshitgit (master)$ git pull --rebase
First, rewinding head to replay your work on top of it...
Applying: Meaningful commit message
git push

ksylormiller:~/ohshitgit (master)$ git push
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 331 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/ksylor/fluent-example.git
  f6e51fd..3c2335c  master -> master
Remote/Origin

Github

Local

Your machine

Staging/Index

Workspace

Master

ORIGIN/
MASTER

Master
Viewing and changing history

Viewing history
Changing history
Viewing and changing history

Viewing history

Changing history
View the history of a branch
ksylormiller:~/ohshitgit (master)$ git log
commit 3c2335c50a1ad48a283a475e3d3f9ac445a20ff5
Author: Katie Sylor-Miller <katie@ohshitgit.com>
Date:   Thu Jun 8 23:42:05 2017 -0400

    Another awesome commit

    A detailed message about this commit goes here

commit f6e51fdd6650c7bbc39895c0392ec6a9fb5a9c15
Author: Someone else <someone@ohshitgit.com>
Date:   Thu Jun 8 23:39:36 2017 -0400

    Oh hai this is a change

commit 4f9a80ff7732f252ffbe7841e96748707da1a78d
git reflog is your magic time machine

by Etsy seller Birdwoodshack
ksylormiller:$~/ohshitgit (master)$ git reflog

3c2335c HEAD@{0}: rebase finished: returning to refs/heads/master
3c2335c HEAD@{1}: pull --rebase: lolz
f6e51fd HEAD@{2}: pull --rebase: checkout f6e51fdd6650c7bbc39895c0392ec6a9fb5a9c15
f4f4d99 HEAD@{3}: reset: moving to HEAD@{4}
3c2335c HEAD@{5}: reset: moving to HEAD~
1ecad3a HEAD@{6}: rebase finished: returning to refs/heads/master
1ecad3a HEAD@{7}: rebase: lollz
f6e51fd HEAD@{7}: rebase: checkout origin/master
f4f4d99 HEAD@{8}: commit: lolz
4f9a80f HEAD@{9}: commit: Another awesome commit
4875652 HEAD@{10}: commit: Meaningful commit message
HEAD notation

- HEAD@{index} : go to a specific point
- HEAD~ == HEAD@{currentindex+1} : back one action
- HEAD~x == HEAD@{currentindex+x} : back x actions
- Or, you can always reference by hash
ksylormiller:~/ohshitgit (master)$ git show HEAD@{8}
commit f4f4d996d44bbf824cdd22ace9e6d154d27c9a5d
Author: Katie Sylor-Miller <katie@ohshitgit.com>
Date:   Thu Jun 8 23:42:05 2017 -0400

lolz

diff --git a/bar.php b/bar.php
index f81fce0..bddd3cb3 100644
--- a/bar.php
+++ b/bar.php
@@ -1 +1,4 @@
 this is a file
 -with nothing
 +with some lolz
git checkout

ksylormiller:~/ohshitgit (master)$ git checkout HEAD@{8}
Note: checking out 'HEAD@{8}'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b new_branch_name

HEAD is now at a4df41a... Merge remote-tracking branch 'origin/master'
ksylormiller:~/ohshitgit (master) ((detached from a4df41a))$
A detached HEAD is when your HEAD points directly to a commit or a tracking branch (not an editable branch)
post-checkout detached HEAD

6dab6a7 B

aec5611 C

D

912bde5 E

1668d2f F

master

HEAD
Changing history of a branch
git reset brings you back to a previous state in history, undoing everything that changed between now and then

Hit the reset button
git reflog

ksylormiller:~/ohshitgit (master)$ git reflog
1668d2f HEAD@{0}: rebase finished: returning to refs/heads/master
3c2335c HEAD@{1}: pull --rebase: lolz
912bde5 HEAD@{2}: pull --rebase: checkout
f6e51fdd6650c7bbc39895c0392ec6a9fb5a9c15
f4f4d99 HEAD@{3}: commit: lolz
ksylormiller:~/ohshitgit (master)$ git reset HEAD@{3}
Unstaged changes after reset:
M foo.php
## Three basic types of resetting

<table>
<thead>
<tr>
<th>--soft</th>
<th>--mixed</th>
<th>--hard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Leaves the changes from the affected commits in staging</td>
<td>Leaves the changes from the affected commits in the workspace (default)</td>
<td>Discards the changes from the affected commits</td>
</tr>
</tbody>
</table>
Changes move to the index or workspace or discarded
Disclaimer: never change public history
git revert creates a new commit that undoes the changes in the specified commit(s)
ksylormiller:~/ohshitgit (master)$ git revert f4f4d99
ugh, vim again

Revert "lolz"

This reverts commit f4f4d996d44bfb824cdd22ace9e6d154d27c9a5d.

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Your branch is ahead of 'origin/master' by 3 commits.
# (use "git push" to publish your local commits)
#
# Changes to be committed:
#   modified:   bar.php
#
~
"~/ohshitgit/.git/COMMIT_EDITMSG" 14L, 450C
ksylormiller:~/ohshitgit (master)$ git revert
f4f4d996d44bbf824cdd22ace9e6d154d27c9a5d
[master b483271] Revert "lolz"
  1 file changed, 3 deletions(−)
Post-revert

reverts changes from G
Avoiding & getting out of messes

Always be committing
Always be branching
Avoiding & getting out of messes

Always be committing

Always be branching
Lots of commits are better

- Commits are cheap and easy
- Save your progress over time
- Easier to backtrack if you screw up
- Less chance of committing the wrong thing when you are reviewing small changes
Problem: Committing the wrong thing
Avoid the problem: Committing the wrong thing

- Set up your command line to show you what branch you are on
- Use .gitignore to your advantage
- Remember git status is your BFF
- Understand how staging works (subsequent changes to a file are not reflected)
Fix the problem: Entering the wrong commit message

ksylormiller:~/ohshitgit (master)$ git commit --amend
# insert vim here
Fix the problem: Entering the wrong commit message

ksylormiller:~/ohshitgit (master)$ git commit --amend
[master f0c2e62] My correct commit message
1 file changed, 3 deletions(−)
Fix the problem: Forgetting to commit a file

```bash
ksylormiller:~/ohshitgit (master)$ git add file/i/forgot/to/commit.ext
ksylormiller:~/ohshitgit (master)$ git commit --amend --no-edit
[master 014035a] My correct commit message
  1 file changed, 1 insertion(+), 3 deletions(-)
```
Avoiding & getting out of messes

Always be committing

Always be branching
Why a feature branch workflow is better

- Safety net: you aren’t changing master directly until your feature is ready for prime time
- Allows you to switch between tasks/manage unrelated changes
- Share larger features or long-term work with your team and preserve its history
- Caution: Don’t use *long-lived* feature branches
Problem: Staying up to date
Merge vs. rebase

**MERGE**

Adds a new commit to your feature branch.

You are resolving potential conflicts created by other people's code.

**REBASE**

Replays your commits on top of the latest of master.

You are resolving potential conflicts created by your own code.
Problem: Rebasing kinda sucks
Avoid the problem: Combine commits before rebasing

ksylormiller:~/ohshitgit (feature-branch)$ git rebase -i HEAD~2
Interactive rebase

pick 2f5bae4 example feature branch
pick 0ee2a20 Another commit

# Rebase aaff258..0ee2a20 onto aaff258
#
# 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
~
"~/ohshitgit/.git/rebase-merge/git-rebase-todo" 20L, 670C
Interactive rebase

pick 2f5bae4 example feature branch
fixup 0ee2a20 Another commit

# Rebase aaff258..0ee2a20 onto aaff258
#
# 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
~
"~/ohshitgit/.git/rebase-merge/git-rebase-todo" 20L, 670C
Avoid rebasing problems: combine commits

ksylormiller:~/ohshitgit (feature-branch)$ git rebase -i HEAD~2
[detached HEAD a682e3a] example feature branch
  3 files changed, 5 insertions(+)
Successfully rebased and updated refs/heads/feature-branch.
Problem: Merging back into master
Add more commits by pushing to the **feature-branch** branch on ksylor/fluent-example.

This branch has no conflicts with the base branch
Merging can be performed automatically.

- **Merge pull request**
- **Create a merge commit**
  - All commits from this branch will be added to the base branch via a merge commit.
- **Squash and merge**
  - The 2 commits from this branch will be combined into one commit in the base branch.
- **Rebase and merge**
  - The 2 commits from this branch will be rebased and added to the base branch.

You can also open this in GitHub Desktop or view command line instructions.
Problem: Merge conflicts
Avoid the problem: Merge conflicts

- Update your local master and origin/master all.the.time.
- Rebase your feature branches to stay up to date
- Periodically squash feature branch commits to reduce the number of conflicts to resolve (esp. when rebasing)
- Communication with teammates
- If things don’t seem right, abort!
this is my worst nightmare
check your diffs before committing!

ksylormiller:~/ohshitgit (master)$ git diff --check
foo.php:1: leftover conflict marker
foo.php:4: leftover conflict marker
foo.php:5: trailing whitespace.
+fdfakadsfldjfasd^M
foo.php:6: leftover conflict marker
That’s it! Easy, right?
Avoid git messes in the first place

• Understand the fundamentals (commits, branches, HEAD, history, workflows)

• Know how to view & fix history (reflog is git’s magic time machine)

• Always be committing

• Always be branching (but stay up to date)

• DON’T PANIC, everything is fixable
If all else fails
ohshitgit.com
Thank you!

@ksylor
@ohshitgit