Oh-My-Git

Ashik Salman

Awesome Git tweaks

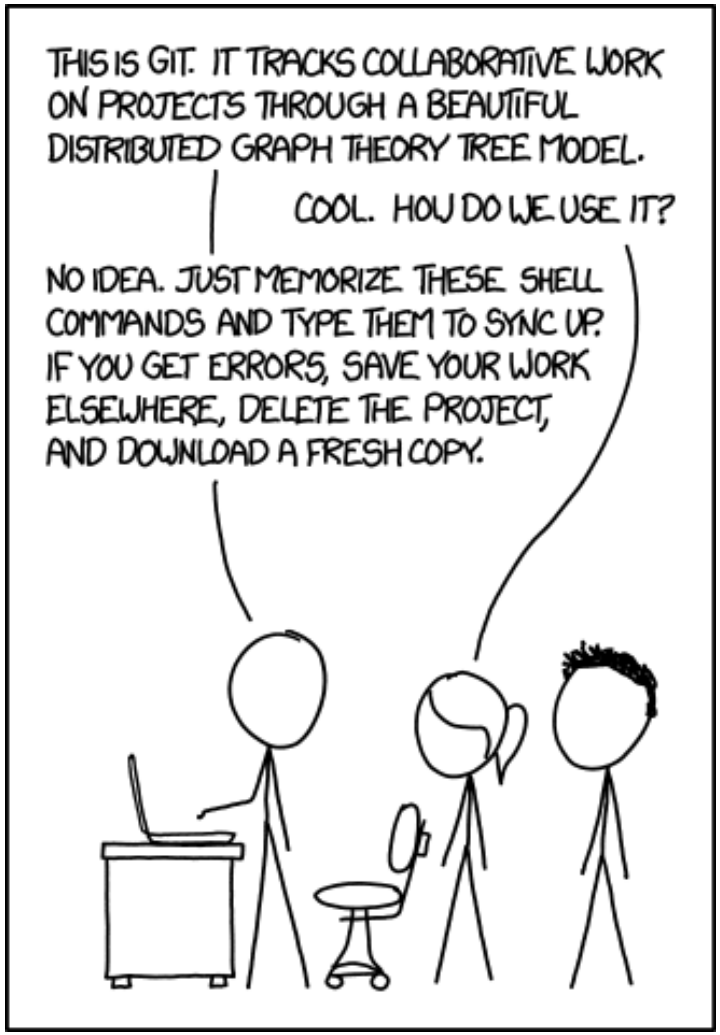
.....

Backend Developer
Chillr, Backwater Technologies
Kochi, Kerala

September 25, 2016

oh-my-git, what?

Git Tool


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

Versioning by Patch changes

Git with Patches

Add changes by patches

```
$ git add -p
$ git add -i (interactive menu)
```

Git with Patches

Add changes by patches

```
$ git add -p
$ git add -i (interactive menu)
```

Checkout changes by patches(Remove un-staged changes)

```
$ git checkout -p
```

Git with Patches

Add changes by patches

```
$ git add -p
$ git add -i (interactive menu)
```

Checkout changes by patches(Remove un-staged changes)

```
$ git checkout -p
```

Reset changes by patches(Remove staged changes)

```
$ git reset -p
```

oh-my-git !!!

Scenario 2

I just committed some changes and immediately realized I need to make one small change.

oh-my-git !!!

Example (Solution)

```
# make your change
$ git add .  # or add individual files
$ git commit --amend
# follow prompts to change or keep the commit message
# now your last commit contains that change!
```

oh-my-git !!!

Example (Solution)

```
# make your change
$ git add .  # or add individual files
$ git commit --amend
# follow prompts to change or keep the commit message
# now your last commit contains that change!
```

Note !

You could also make the change as a new commit and then do rebase -i in order to squash them both together, but this is about a million times faster.

oh-my-git !!!

Scenario 3

I accidentally committed something to master that should have been on a brand new branch!

Example (Solution)

```
# create a new branch from the current state of master
$ git branch some-new-branch-name
# remove the commit from the master branch
$ git reset HEAD~ --hard
$ git checkout some-new-branch-name
# your commit lives in this branch now :)
```

Example (Solution)

```
# create a new branch from the current state of master
$ git branch some-new-branch-name
# remove the commit from the master branch
$ git reset HEAD~ --hard
$ git checkout some-new-branch-name
# your commit lives in this branch now :)
```

Note !

This doesn't work if you've already pushed to origin, and if you tried other things first, you might need to git reset HEAD@number instead of HEAD .

oh-my-git !!!

Scenario 4

Ooops..! I accidentally committed to the wrong branch!

oh-my-git !!!

Example (Solution)

```
# undo the last commit, but leave the changes available
$ git reset HEAD~ --soft
$ git stash
# move to the correct branch
$ git checkout name-of-the-correct-branch
$ git stash pop
$ git add . # or add individual files
$ git commit -m "your message here"
# now your changes are on the correct branch
```

Note !

A lot of people have suggested using cherry-pick for this situation too, so take your pick on whatever one makes the most sense to you!

Note !

A lot of people have suggested using cherry-pick for this situation too, so take your pick on whatever one makes the most sense to you!

Example (Solution)

```
$ git checkout name-of-the-correct-branch
# grab the last commit to master
$ git cherry-pick master
# delete it from master
$ git checkout master
$ git reset HEAD~ --hard
```

oh-my-git !!!

Scenario 1

I did something terribly wrong, I want to got to a stage where everything worked fine.

oh-my-git !!!

Scenario 1

I did something terribly wrong, I want to got to a stage where everything worked fine.

Why... ?

You can use this to get back stuff you accidentally deleted, or just to remove some stuff you tried that broke the repo, or to recover after a bad merge, or just to go back to a time when things actually worked.

oh-my-git !!!

Example (Solution)

```
$ git reflog
# you will see a list of every thing you've done in
# git, across all branches!
# each one has an index HEAD@{index}
# find the one before you broke everything
$ git reset HEAD@{index}
```

Example (Solution)

```
$ git reflog
# you will see a list of every thing you've done in
# git, across all branches!
# each one has an index HEAD@{index}
# find the one before you broke everything
$ git reset HEAD@{index}
```

Note !

If the work was committed at any point, then it can be recovered from the reflog. By default all commits stay alive in the reflog for at least 2 weeks.

oh-my-git !!!

Total mess-up !

```
# Suppose only one branch went terribly wrong...!
$ git checkout branch-name
$ git reset HEAD~50 --hard
$ git pull upstream branch-name
$ git push origin -f branch-name

# Delete the repo and clone a fresh one.
$ cd ..
$ sudo rm -r messed-git-repo-dir
$ git clone
$  https://some.github.url/messed-git-repo-dir.git
$ cd messed-git-repo-dir
```

Git Merge (Conflicts!!!)

oh-my-git !!!

Basic Merge !

```
$ git checkout master
# Switched to branch 'master'
$ git merge iss53
# Merge made by the 'recursive' strategy.
# index.html |    1 +
# 1 file changed, 1 insertion(+)

$ git fetch & git merge == git pull
```

Basic Merge conflicts

```
$ git merge feature-branch
# Auto-merging index.html
# CONFLICT (content): Merge conflict in index.html
# Automatic merge failed; fix conflicts and then
# commit the result.

# <<<<<<< HEAD:index.html
# <div id="footer">contact : email.support@github.com</div>
# =======
# <div id="footer">
#  please contact us at support@github.com
# </div>
# >>>>>>> feature-branch:index.html
```

oh-my-git !!!

Abort merge

```
$ git merge --abort
```

oh-my-git !!!

Abort merge

```
$ git merge --abort
```

Fix files

```
$ Fix conflicts manually
$ git add files
$ git commit
```

oh-my-git !!!

Abort merge

```
$ git merge --abort
```

Fix files

```
$ Fix conflicts manually
$ git add files
$ git commit
```

Checkout files

```
# Suppose you have conflict with a file
# which doesn't have any of your code in it.
$ git checkout master -- path/to/file-name
```

Git rebase - interactive (Conflicts !!!)

Git Rebase

The Perils of Rebasing

- Do not rebase commits that exist outside your repository.
- Abandoning existing commits and creating new ones that are similar but different.
- Collaborators will have to re-merge their work and things will get messy (If pushed to remote)

Git Stash - Easy work save/restore

Thank You

Questions?