Git Tutorial

André Sailer

CERN-EP-LCD

DIRAC User Workshop
Montpellier, May 25, 2016

Table of Contents

Introduction, Caveat

- Assuming you know what git is for
- Assuming you have at least very some basic knowledge of git
  - clone, fetch, pull, push, commit
- Giving my opinion about useful commands and workflows
- Keeping the history reasonably clean
- log messages are part of the *documentation*
  - Why was this line with a not so obvious bug last changed? `git annotate` $\rightarrowtail$ "insert your favourite useless commit message" $\rightarrowtail$ #$@%$

Most importing thing about git: You can undo (almost) everything
Commit early and often and later re-write the history

Section 1:

1 Introductory Remarks

Tutorial repository:

- I assume you have some kind of git command line interface (linux, mac?, cygwin??)
- Make a clone of the tutorial repository which contains many branches for the exercises
  `git clone https://github.com/andresailer/tutorial.git`
- Go to the exercise branch:
  `git checkout exercise1`
- Follow along with the commands: *git is easy!*
- Not necessary to remember how everything can be done, just that there is a way to do it. Just google it

Some useful things for git

- bash completion for git
  - completes: commands, options, branches, tags, remotes
  - Keeps one from having to type so much
  - and easier to remember (`--amend`, `--dry-run`, `rebase --interactive`, ...)
- git status in terminal (bash)

  `sailer@localhost:~/Work/TalksGit/160525_DiracWS_GitTutorial(master*+)$`
  - on branch master
  - unstaged changes *
  - staged changes +
- alias to see the history of branches and commits (add to your /.gitconfig
  - `lola = log --graph --decorate=full\n --pretty=oneline --abbrev-commit --all`

```
* 73756b8 (refs/remotes/origin/integration, refs/remotes/origin/HEAD) Avoid twice the Travis icon
*   4e5969e Merge branch 'rel-v6r15' into integration
|\
* \   966c351 Merge branch 'rel-v6r15' into integration
|\ \
* | | 77ca9f0 use WebAppDIRAC v1r6p29
* | | 18c1386 v6r14p31, v6r15
* | |   756c36e Merge branch 'rel-v6r15' into integration
|\ \ \
* | | | 0c8b4cc v6r15-pre24
* | | |   1a75a90 Merge branch 'rel-v6r15' into integration
|\ \ \ \
* | | | | 7518ecb typo
* | | | |   f272a68 Merge branch 'rel-v6r15' into integration
|\ \ \ \ \
* | | | | | f0b4330 v6r14p30, v6r15-pre23
* | | | | |   7959014 Merge branch 'rel-v6r15' into integration
|\ \ \ \ \ \
* | | | | | | ec0a7fe Use WebAppDIRAC v1r6p27
* | | | | | |   9752287 Merge branch 'rel-v6r15' into integration
```

Section 2:

2 Committing

Exercise0: Status, Staging and Committing

- After cloning the tutorial repository from github you should be on the master branch
- lets create a file `touch someNewFile`
- See the status: `git status`
  - New file is now *untracked*

```
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       someNewFile
nothing added to commit but untracked files present (use "git add" to track)
sailer@localhost:~/Work/Gittutorial (master)$
```

- `git add someNewFile`
  - the file is now *staged* and will be added to the new commit

```
sailer@localhost:~/Work/Gittutorial (master +)$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       new file:   someNewFile
#
sailer@localhost:~/Work/Gittutorial (master +)$
```

- `git commit -m"proper commit message"`

Exercise1: Control what is going to be committed

- `git checkout exercise1`
- Undo the last commit from branch exercise1: `git reset HEAD~`
- chose which line to add: `git add -p`
  and *follow the instructions* git gives:
  1. press “s” to split the possible changes into smaller slices
  2. accept the first one: “y”
  3. stop now: “q” to quit or “n” to reject this one for now
  4. Look at the status of the repository: `git status`

Exercise1: Part Staged, Part Not

```
sailer@localhost:~/Work/gitclone/tutorial (exercise1 *+)$ git st
# On branch exercise1
# Your branch is behind 'origin/exercise1' by 1 commit, and can be fast-forwarded.
#   (use "git pull" to update your local branch)
#
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   exercise1/exercise1.txt
#
# 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:   exercise1/exercise1.txt
#
sailer@localhost:~/Work/gitclone/tutorial (exercise1 *+)$
```

- We staged only part of the changes in this file.
- The rest can be committed later, or discarded

Exercise1: Commit the rest

- Lets continue committing this change:
  1. to see the *staged* changes: `git diff --staged`
  2. See *unstaged* changes: `git diff`
  3. commit : `git commit -m"add line 2"`
- Now repeat this for the next line, avoid committing the debug statement

Why not just `git commit -a -m"fix"`? Group code changes; avoid committing debug statements; *review your own code before committing*

Exercise2: More commitment

Oops, I forgot to add one thing; have to fix a typo; ...

- add to the last commit: `git commit --amend`
  1. go to exercise 2: `git checkout exercise2`
  2. open the file and fix the typo in line 3
  3. Staged the change: `git add -p`
  4. update the last commit: `git commit --amend`
     - ⋆ this also opens the editor to change the last commit message
     - ⋆ to only change the last commit message call `git commit --amend` without staged changes
     - ⋆ To just commit without changing the commit message: `git commit --amend --no-edit`

Why?: Avoiding small commits with messages like “typo”, “fix”,“temp”,. . . , fewer commits, but more sensible ones, you will be happy if you have to rebase later on and not fix the same line three times

Section 3:

3 Re-writing history

Exercise3: Interactive Rebase 1

Rewriting history for a more sensible order: Mostly will be used in conjunction with squashing commits, joining two commits that belong together similar to amending commits

- go to exercise 3: `git checkout exercise3`
- See the two commits that are not in the proper order, we want to count to 8 before we count to 12 obviously...
- To change the order:
  1. `git rebase --interactive HEAD~3`
  2. This opens an editor with the list of commits and instructions how to proceed

★ the oldest commit is on top

★ changing the order of the lines will change the order of commits

★ removing a line will remove the commit

★ just the first word needs to be changed, one letter is enough

```
1pick 8c05ad6 Start with exercise 3: count to 4
2pick d37e2b0 Exercise 3: count to 12
3pick c3e7392 Exercise 3: count to 8
4
5# Rebase 700ed26..c3e7392 onto 700ed26
```

3. Change the order of the second and third line, save the file and exit the editor
4. Look at the commit history again

Exercise4: Interactive Rebase 2

What if we want to change a commit before the last one?

- go to exercise 4: `git checkout exercise4`
  1. See in the history: the third commit fixes a typo belonging to the first commit
  2. See the first commit: *git show HEAD~2*
  3. See the third commit: *git show HEAD*
  4. Using rebase interactive we can change the order and *squash* or *fixup* the third commit into the first
     1. Start the rebase: `git rebase --interactive HEAD~3`
     2. Change the order and change the first word of the now second commit to *squash*, which will join the two commits and open an editor asking you to do something with the two commit messages
     3. See the new commit: *git show HEAD~*

Nota Bene

- The oldest commit is on top (I know I said this before)
- You can use the commit sha in the rebase command, but you need the one before the last commit you want to change: e.g, `00d570c~`
- You can also just change commit messages: `reword`
- Join commits and ignore their log message: `fixup`
  - Ideal to get rid of those *temp* commits
  - Fixup/squash can also be done controlled directly via the commit message
- Do all of these things in one go of interactive rebase
- rebasing like this *can* lead to conflicts

Section 4:

4 Rebasing and Merging

Exercise: Rebasing local branches

- All exercise branches are started from the master branch, lets grow the master branch beyond the exercise branches

```
1 git checkout master
2 touch newfile
3 git add newfile
4 git commit -m"newfile"
```

- Now the master branch has a commit beyond the exercises, if we want our exercise branches to catch up the master we can either `merge` the master into the exercise, or `rebase` the exercise back to the master
- Let's do both

```
1 git checkout exercise3
2 git merge master
```

```
1 git checkout exercise4
2 git rebase master
```

Rebase vs. merge of local branches

```
sailer@localhost:~/Work/gitclone/tutorial (exercise4)$ git lola
* a72266e (HEAD, refs/heads/exercise4) correct typo in line3
* 6a9070b Exercise 4: count to 8
* 0992ccd Start with exercise4 and count to 4
| *   b79754d (refs/heads/exercise3) Merge branch 'master' into exercise3
| |\
| |/
|/|
* | 4c26167 (refs/heads/master) newfile
| * c3e7392 (refs/remotes/origin/exercise3) Exercise 3: count to 8
| * d37e2b0 Exercise 3: count to 12
| * 8c05ad6 Start with exercise 3: count to 4
|/
| * 59de3b2 (refs/remotes/origin/exercise4) correct typo in line3
| * 00ef069 Exercise 4: count to 8
| * 00d570c Start with exercise4 and count to 4
|/
| * 3de36d1 (refs/remotes/origin/exercise1, refs/heads/exercise1) Add line 2 and line 5 and debug to be removed
| * 17f5326 Start with exercise1
|/
| * b54f7ef (refs/remotes/origin/exercise2, refs/heads/exercise2) Add exercise2: counting lines
|/
* 700ed26 (refs/remotes/origin/master, refs/remotes/origin/HEAD) update outline
* 9b8dbf8 Add tutorial outline
sailer@localhost:~/Work/gitclone/tutorial (exercise4)$
```

- Rebase: “moves” the commits from the branch to start from a different commit
  - One can rebase from and to pretty much anything
- Merge: Merges the two branches, signified by some commit
  - I tried to avoid them as much as possible
  - Why do I care when you merged something else into your branch?
  - Even worse when pulling from remotes and merging . . .

Another Caveat

- Rewriting history is dangerous
- Only re-write history that has not been shared with others
- As long as your pull request has not been merged, it is OK
- To push to your own remote repo use `git push -f`
- Github/Gitlab will update your PR with the forced branch

Exercise: Pull – Merge vs. Rebase

Let's update from a remote via somewhat constructed examples:

- `git checkout exercisePull1`
  1. Look someone made changes in the upstream...
  2. we want to incorporate those changes into our development
  3. `git pull --no-rebase origin exercisePull2`
  4. No merge, because we could *fast-forward* our branch to the remote changes. The upstream commits were simply added after our own.
- Now, what if this was a parallel development from a different branch
  1. `git pull --no-rebase origin exercisePull3`
  2. pulling like this creates a merge commit

Section 5:

5 Undoing: Reflog

Exercise: Reflog

- So clearly we did not want to do that pull without rebase, so what do we do?
- We can undo it
  1. List the repository status: `git reflog`
  2. Find the one before the last `git pull`: `HEAD@{1}`
  3. Reset the repository to this state: `git reset HEAD@{1}`
- You can also undo that undoing, see `git reflog` again

Section 6:

6 More Pulling

Exercise: git pull with rebase

- Now lets pull this other branch again
  1. to be sure of the branch: `git checkout exercisePull1`
  2. now pull with rebase: `git pull --rebase origin exercisePull3`
- one can set rebase to the default in `.gitconfig`

```
[branch]
        autosetuprebase = always
```

Exercise: Conflict resolution

What happens when the upstream changes conflict with our local ones?

- New exercise branch: `git checkout exerciseConflict1`
- Pull with merge and conflict
  - `git pull --no-rebase origin exerciseConflict2`
  - Merge conflict that needs to be fixed

Let's see what happens when we do a rebase:

- reset the branch: `git reset --hard origin/exerciseConflict1`
  1. Pull with rebase: `git pull --rebase origin exerciseConflict2`
  2. Conflicts can (have to?) be treated one at a time:
  3. Fix conflict, stage file and then
     - ★ either `git rebase --continue`
     - ★ or `git rebase --skip` if the change was made obsolete by the upstream changes

Section 7:

7 Git stash

stashing uncommitted changes

- To rebase, pull, etc. one needs a clean working directory without any uncommitted changes.
- What to do when one does not want to commit them (e.g., just debug printouts statements)?
- Create a stash of your changes in tracked files: `git stash save ["debug prints"]`
- Do you pull/rebase/etc.
- Apply the stashed changes to the current head again: `git stash pop`
- See the list of stashes and the repository state they belong to: `git stash list`
- Drop stashes you no longer need: `git stash drop stash@{xyz}`

EOF