Git Tutorial

André Sailer

CERN-EP-LCD

August 18, 2016

Table of Contents

1 Introduction
- Setup
- Some useful things for git

2 Basic git commands
- Creating a Repository
- Committing
- Branches

3 More Advanced Change Control
- Exercise 1: More fine grained control over commits
- Exercise 2: Amending Commits
- Exercise 3: Re-writing history
- Exercise 4: Re-writing History Again

4 Rebasing and Merging

5 Undoing: Reflog

6 More Pulling

7 Git stash

8 More Real Life Example and Collaborating via git(lab)
- Merge Request

9 Summary

10 Further Reading

Section 1:

1 Introduction
- Setup
- Some useful things for git

Introduction

- What is *git* for
  - Keep track of changes in the source code
  - Either for one person or collaborative work
  - Similar to SVN, but the *how* of what git is doing is completely different, so I will not mention SVN again

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

- Not necessary to remember how everything can be done, just that there is a way to do it. Just google it

Setup

Git provided on clicdp CVMFS

- Up-to-date version: 2.9.2 (as opposed to 1.7.1 or 1.8.3.1)
- Includes git-completion and git-prompt

To enable it on any computer with CVMFS installed

```
source /cvmfs/clicdp.cern.ch/software/git/2.9.2/x86_64-slc6-gcc48-opt/setup.sh
```

You can (and should later) also add this line to your `${HOME}/.bashrc` file

command completion for git

bash completion for git

- For example: type `git commit --<tab>` to get all options for commit
- completes: commands, options, branches, tags, remotes
- Keeps one from having to type so much
- and easier to remember (`--amend`, `--dry-run`, `rebase --interactive`, ...)

Maybe you also have to add

```
if [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
fi
```

to your `${HOME}/.bashrc` file, somewhere before sourcing the git setup mentioned above.

Git status in the terminal

Keep track of the current branch you are working on and if there

- git status in terminal (bash)

```
sailer@localhost:~/Work/TalksGit/160525_DiracWS_GitTutorial(master*+)$
```

  - on branch master
  - unstaged changes *
  - staged changes +
  - rebase in progress

Displaying History

To better understand where in the tree of commits one is located

```
* 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
|\ \ \ \ \ \ \
* | | | | | | | b6ad579 Removed renamed files
* | | | | | | | 8ae6d8d Typo
* | | | | | | | ca7e1ff Use VMDIRAC v2r0
* | | | | | | | 3574756 v6r14p28, v6r15-pre21
* | | | | | | |   aea979d Conflicts resolved with rel-v6r15
|\ \ \ \ \ \ \ \
* | | | | | | | | 8a72d00 Removing tests with lower case T
* | | | | | | | |   f781eec Merge branch 'rel-v6r15' into integration
|\ \ \ \ \ \ \ \ \
```

- alias

```
lola = log --graph --decorate=full\n --pretty=oneline --abbrev-commit --all
```

to add to the ${HOME}/.gitconfig file

My ${HOME}/.gitconfig File

```
[user]
        name = Andre Sailer
        email = andre.philippe.sailer@cern.ch
[alias]
        st = status
        ci = commit
        co = checkout
        lola = log --graph --decorate=full\n--pretty=oneline --abbrev-commit --all
        logf = "!echo \"Remember to add -S<string>\" ; git log --color -p --source --all"
[color]
        ui = true
        diff = auto
        status = auto
[core]
        excludesfile = /dev/null
        editor = emacs -nw
[branch]
        autosetuprebase = always
[push]
        default = simple
```

Obviously modify the `name` and `email` and the global `excludesfile`, add an alias for commands you find useful

Section 2:

2 Basic git commands
- Creating a Repository
- Committing
- Branches

Create your own repository

It is very easy to create a local git repository

```
mkdir newFolder
cd newFolder
git init
```

- Now you can keep track of the files in this folder
- Can also be done in an existing directory with your super important scripts that you never thought about putting into version control

See Status of working directory

See the current status of the working directory: files with changes, *untracked* files, *staged* files

```
git status
```

```
On branch master
Initial commit
nothing to commit (create/copy files and use "git add" to track)
```

At the moment there is nothing, but *git* is telling you what you can do next.

Commit a file

Let us add a file to this repository we just created

```
touch someNewFile
git status
```

```
On branch master
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be commited)

        someNewFile

nothing added to commit but untracked files present
 (use "git add" to track)
```

here we see we have an *Untracked* file

Commit a file II

Let us tell git to start keeping track of the file

```
git add someNewFile
git status
```

```
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   someNewFile
```

Now we have a *Staged* file, the next time we do `git commit` the file and its content will be part of the *commit*

Commit a file III

```
git commit --message "add someNewFile with important information"
```

```
[master (root-commit) ff9ea62] add someNewFile
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 someNewFile
```

```
git status
```

```
On branch master
nothing to commit, working tree clean
```

Now we have the first successful addition to our repository

Clone someone else's repository

- Of course we don't always want to start from scratch, so we can also *clone* the repository from somewhere else
- Usually something like *github* or *gitlab*, but you can also use a location in a different folder on a file system like *afs*

Tutorial repository

- Go to a new clean folder
- Make a clone of the tutorial repository which contains many branches for the exercises

  ```
  git clone
  https://:@gitlab.cern.ch:8443/CLICdp/GitTutorial.git
  ```

- See the status of the current repository: Notice that it tells you it is *up-to-date with 'origin/master'*

  ```
  git status
  ```

  ```
  On branch master
  Your branch is up-to-date with 'origin/master'.
  nothing to commit, working tree clean
  ```

- *origin* is *a* remote repository, and our branch is supposed to keep track of changes in that remote branch, so we can easily stay up-to-date with the development of other people

Branches

- Sequences of commits are called a *branch*
- To switch between branches use `git checkout <branch>`
  - Remember to make use of the *completion* of branch names
- Every branch is as important as any other, only by convention are some branches considered more important, e.g., *master*
- Let's go to a different branch in this repository `git checkout exercise1`
  - This switches to one of the branches cloned from the remote repository
- See the status of the branches `git branch -vv`

```
* exercise1 3de36d1 [origin/exercise1] Add line 2 and line 5
  master    8a331eb [origin/master] Include images for PR
```

We are on branch *exercise1*, which is at the commit *3de36d1*, is tracking the remote branch *origin/exercise1* and at the end is the beginning of the commit message

Section 3:

3 More Advanced Change Control

- Exercise 1: More fine grained control over commits
- Exercise 2: Amending Commits
- Exercise 3: Re-writing history
- Exercise 4: Re-writing History Again

Exercise 1: Control what is going to be committed

When we don't add a completely new file, we should take care to only commit changes that we actually want to commit, for this we can chose which changes from a file we want to *stage* for the next commit

- `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`

Exercise 1: 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
- Note: again *git* is telling you what you can do and how to do it

Exercise 1: 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*

Exercise 1: Summary

- Check the status of the working directory, the line starting with *BLABLABLA* should not be committed
- `git diff` should show only this line as difference
- `git show HEAD` will show you the last line
- `git lola` shows you were your branch is now with respect to the other branches
- `git branch -vv` tells you the difference with the remote branch
  - Because of the `git reset` we did above the are *behind* the remote branch
- Discard the change we don't want to keep:
  `git checkout exercise1/exercise1.txt`
  - yes, the same `checkout` that switches between branches is also used to change files this is one of the dangerous things that can lead to loss of work

Exercise 2: Amending Commits

When we commit early and often we might forget something:
Oops, I forgot to add one thing; have to fix a typo; . . .
We don't have to create a completely now commit just to fix our mistake, we can modify the existing one

- To 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

Exercise 3: 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 (e.g., via *git lola*) , 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: Re-writing History Again

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 try 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`
  `git branch --set-upstream-to origin/exercisePull2`
  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
git branch --set-upstream-to=origin/exerciseConflict2
```

- 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}`

Section 8:

8 More Real Life Example and Collaborating via git(lab)

- Merge Request

Collaborating: Forks and Multiple Remotes

1. Clone the tutorial document: `git clone https://:@gitlab.cern.ch:8443/CLICdp/TutorialDocument.git`
2. go to `https://gitlab.cern.ch/CLICdp/TutorialDocument/forks/new` and for the document
3. add your fork as a second remote to your local working directory:

```
git remote add downstream https://:@gitlab.cern.ch:8443/<UserName>/TutorialDocument.git
```

- you can also copy the URL from the gitlab webpage
- *downstream* is just a name for the remote repository
- You can have many different remotes on different servers, e.g., also from github, at the same time

Obtaining upstream changes

- To obtain changes from the remote repositories use `git fetch <remote>`
- This will *not* apply changes to your local branches

Creating a Merge Request I

1. Edit any file you want, commit and push your changes
   1. Create a new branch: `git checkout -b newBranch`
   2. edit a file
   3. commit your changes:

      ```
      git add -p
      git commit -m"nice description"
      ```
   4. Push your changes to the *downstream* remote: `git push downstream myNewBranch`
   5. Create a *Merge Request* on your page: `https://gitlab.cern.ch/<userName>/TutorialDocument`

Creating a Merge Request II

Andre Sailer / TutorialDocument · GitLab - Mozilla Firefox

Andre Sailer / TutorialDocument

Project Activity Repository Pipelines Graphs Issues Merge Requests 0

You pushed to **newBranch** branch 4 minutes ago

Create Merge Request

New Merge Request

TutorialDocument

Forked from CLICdp

Star 0 Fork 0 KRBS https://:@gitlab.cern.ch:8443/sailer/TutorialDc Global

Files (110 KB) Commit (1) Branches (2) Tags (0) Add Changelog Add License Add Contribution guide

22751e24 Start with Tutorial Document · 15 minutes ago by Andre Sailer

Push events Merge events Comments Team

Andre Sailer pushed new branch **newBranch** at **Andre Sailer / TutorialDocument** 4 minutes ago
42716ecd · Add section about git stash
Create Merge Request

Andre Sailer created project **Andre Sailer / TutorialDocument** 14 minutes ago

Projects · Todos 0 · Activity · Groups · Milestones · Issues 0 · Merge Requests 1 · Snippets · Help · Profile Settings · sailer

https://gitlab.cern.ch/sailer/TutorialDocument/merge_requests/new?merge_request[source_branch]=newBranch&merge_request[source_project_id]=10581&merge_request[target_branch]=master&merge_request[target_project_id]=10580

Creating a Merge Request III

New Merge Request · Andre Sailer / TutorialDocument · GitLab - Mozilla Firefox

File Edit View History Bookmarks Tools Help

https://gitlab.cern.ch/sailer/TutorialDocument/merge_requests/new?merge_request[source_branch]=newBranch&merge_request[source_project_id]=10581&merge_request

Andre Sailer / TutorialDocument

Project Activity Repository Pipelines Graphs Issues Merge Requests 0

Projects
Todos 0
Activity
Groups
Milestones
Issues 0
Merge Requests 1
Snippets
Help
Profile Settings

Description — Write Preview — Go full screen

bullet list copied straight from the presentation

Styling with Markdown is supported — Attach a file

Assignee: Select assignee — Assign to me

Milestone: Select milestone — Create new milestone

Labels: No labels yet. — Create new label

Source branch: newBranch

Target branch: master — Change branches

Remove source branch when merge request is accepted.

Submit merge request — Cancel

Commits 1 Builds 1 Changes 1

Most recent commits displayed first

17 Aug, 2016 1 commit

sailer

Commenting on the Merge Request

- Now everyone (with the proper access rights) can see and comment on the changes
- Additional commits can be pushed to the Merge Request

Continuous Integration

- As you notice: The changes from your merge request are picked up and already build into a pdf document
- If the build fails you can see it and fix your changes
- For every push the build is run again

Section 9:

9 Summary

Summary

Dealing with remotes

- `git clone` *URL*
- `git remote add` *Remote URL*
- `git fetch` *Remote*
- `git pull [`*Remote Branch*`]`
- `git push [`*Remote Branch*`]`

Branches

- `git branch -vv`
- `git checkout` *Branch*
- `git branch` *NewBranch*
- `git checkout -b` *NewBranch*

Committing

- `git status`
- `git add -p [`*Filename(s)*`]`
- `git diff --staged`
- `git commit -m "Message"`
- `git commit --amend`

Other commands

- `git stash [save, list, pop]`
- `git rebase` *Branch*
- `git merge` *Branch*
- `git lola`

Section 10:

10 Further Reading

Further reading

Very nice presentation:
https://indico.cern.ch/event/288437/
(now protected to cern only?)

EOF