git tutorial

a gentle introduction

Lorenz Walthert (@lorenzwalthert)

March 1, 2018

Today's plan

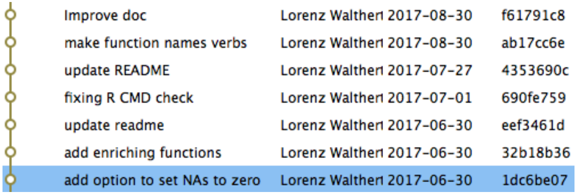
- first part (60 min.): Theory + group practice.
- second part (30 min.): (optional participation): Training problems, general questions.
- Let's make this hands on. Help each other.
- You find these slides here: http://bit.ly/ethz-tutorials -> git tutorial -> pdf

Quick survey

- Who has used RStudio before?
- Who has used git before?
- Who has used the command line before?
- Who has used a text editor like VI, VIM, Emacs?

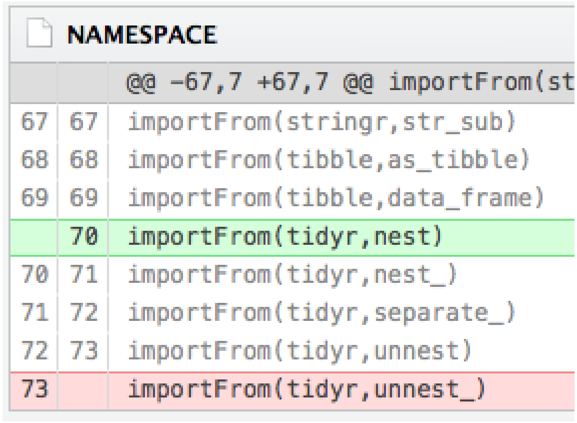
What is git?

- A version control system (VCS).
- smart way of storing file histories.
- Created to version the Linux kernel.
- builds a tree of all changes.
- A change stored in the VCS = a commit.
- Every commit has a hash as an id.



Improve doc	Lorenz Walthert	2017-08-30	f61791c8
make function names verbs	Lorenz Walthert	2017-08-30	ab17cc6e
update README	Lorenz Walthert	2017-07-27	4353690c
fixing R CMD check	Lorenz Walthert	2017-07-01	690fe759
update readme	Lorenz Walthert	2017-06-30	eef3461d
add enriching functions	Lorenz Walthert	2017-06-30	32b18b36
add option to set NAs to zero	Lorenz Walthert	2017-06-30	1dc6be07

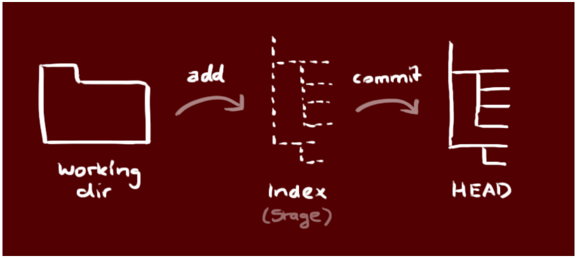
Inspecting differences


NAMESPACE

```
      @@ -67,7 +67,7 @@ importFrom(st
67 67 importFrom(stringr,str_sub)
68 68 importFrom(tibble,as_tibble)
69 69 importFrom(tibble,data_frame)
   70 importFrom(tidyr,nest)
70 71 importFrom(tidyr,nest_)
71 72 importFrom(tidyr,separate_)
72 73 importFrom(tidyr,unnest)
73    importFrom(tidyr,unnest_)
```

Basic workflow

- workflow: change -> save -> add to index -> commit (-> push)
- important: saved not = committed not = pushed

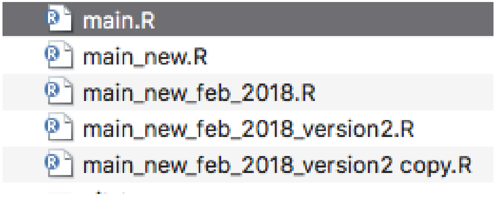


Why git?

Tell me why :-)

Why git?

- end your copy / paste / save code as backup workflow.

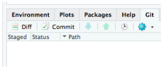


main.R
main_new.R
main_new_feb_2018.R
main_new_feb_2018_version2.R
main_new_feb_2018_version2 copy.R

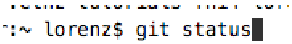
- understand **what** and **why** you did something in retrospect.
- collaborating with others.
- restoring previous versions.
- and more.

How can I use git?

- local GUI (e.g. RStudio git tab).

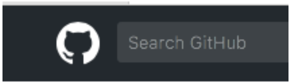


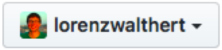
- command line.



```
:~ lorenz$ git status
```

- web GUI (e.g. GitHub).





Pros and Cons

- RStudio git tab: + easy to learn, - limited functionality
- command line: + very flexible, + “native git”, - more difficult to learn than other methods.
- GitHub: + good collaboration / communication, - not actual coding

git with RStudio (create project)

- create new RStudio Project (otherwise you can't use the git GUI)
- check "create git repository"
- call it `ethz-git-with-rstudio`

git with RStudio (create files)

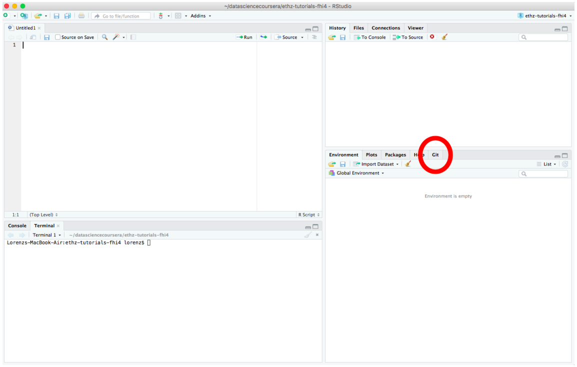
- create a new R script.
- put some example code there.
- I suggest

```
add_two_numbers <- function(x, y) {

sum(x, y)

}

add_numbers(1, 2)
```

- save file as `test-gui.R`

git with RStudio (add files)

- have a look at the git tab



```
Lorenzs-MacBook-Air:ethz-tutorials-fhi4 lorenz$
```

git with RStudio

You have three files there:

- `ethz-git-with-rstudio.Rproj` (project settings)
- `.gitignore`
- `test-gui.R`

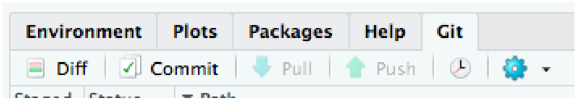
Intermezzo: What is .gitigore?

- A plain text file.
- lists files / folders you want to ignore from being tracked by git, one line per file.
- advanced stuff: removing file from git / complicated inclusion / exclusion rules.

git with RStudio (status of files)

- yellow question mark: untracked by git.
- blue “M”: modified.
- red “D”: deleted.
- not visible: state of file in working dir identical to HEAD

git with RStudio (menu items)



Important for now:

- Diff: compare working dir with HEAD
- Commit: Add changes to tree.
- clock: History (change log).

git with RStudio (adding and committing)

- check boxes of you files.
- Inspect differences with “Diff”.
- Commit them (=add to tree).

Intermezzo: Good commits (messages)

- Do you think “update test-gui.R” is a good commit message? Why / why not?

Intermezzo: Good commits (message)

Example of good commit messages:

- “add missing value analysis to script”
- “remove unused code”

Intermezzo: Good commits (content)

- make small commits.
- changes should be self-contained, minimally complete.
- all changes introduced with a commit should be related to each other.

git with RStudio (adding and committing)

- Review your commit (how?)

git with RStudio

- Improve `add_two_numbers` to ignore missing input values (how?).
- Commit it with a reasonable commit message.
- Again, review your commit (how?).

from GUI to command line

- repeat what we just did on the command line.
- but with a new repo.
- can also mingle GUI and command line approache (that's what happens in practice).
- will learn stuff we could not do with the GUI.

git with the command line (tips)

- use tab to auto-complete file names.
- use star (*) to refer to everyting.
- I use `$` to refer to terminal commands.
- Find doc for git online or `$man git [command]`, e.g. `$man git log`.

git with the command line (intitialization)

- create a new folder `ethz-git-with-cmd` in your home directory.
- navigate there with the command line.
- `$git init`

=> Other ways to create a repo: `$git clone`, see later.

git with the command line (adding files)

- `$touch test-cmd.R` to create empty file.
- add some text & save.
- `$git status`

git with the command line (stage and commit)

- Add file to index with `$git add test-cmd.R`
- could also do `$git add *` to add all.
- check the status (how?).
- `$git commit -m "[commit message here]"`

git with the command line (review commits)

- `$git log`
- `$git log -p`
- `$git log --stat`
- etc.

Commit messages and descriptions

- You now know: `$git commit -m "message"`
- But you can also: `$git commit -m "[message]" -m "[description]"`
- Idea behind description: More verbose and detailed comment on the commit.
- If you need a long (potentially multi-line) description, use a text editor as follows:
- Open text editor for the commit message with `$git commit`

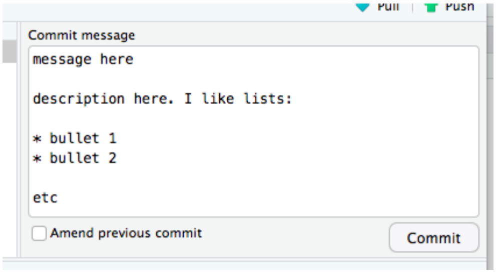
Intermezzo: Using a text editor

- You can now edit the commit message.
- To enter the `insert` mode, press `i`
- First write the commit message, then leave one line blank and write a description.
- When done, hit `ESC`
- Hit `:` (probably `Shift + .` on your keyboard)
- Then, type `wc` (for write and quit).
- Type `cq` if you want to cancel and quit.
- Hit `ENTER` to confirm.

We'll come back to that.

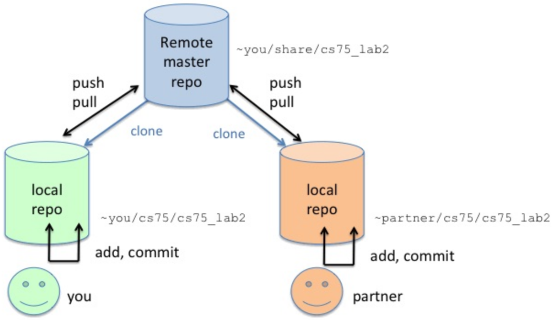
Commit messages and descriptions

- If you are using the RStudio tab, you can also write the message on the first line, then leave one line blank and write the description.



```
message here

description here. I like lists:

* bullet 1
* bullet 2

etc
```

Remote repositories (basics)

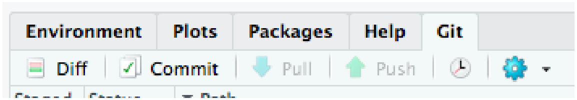


- a remote repo is "just another copy" of a repo (potentially in a different state).
- remote not the same as a branch!
- Never mind about branches we will always use `master`
- Different branches represent different trees.

Remote repositories (basics)

- clone: “Download” a whole repository.
- push: Send your changes to a server.
- pull: “Download” new changes from the server.

git with RStudio (menu items)



New important menu items:

- Pull .
- Push.

But where to push / pull from?

Remote repositories (basics)

Two scenarios:

- You have a repo and want to push it somewhere.
- There is a repo somewhere and you want to pull it.

We'll use GitHub as the server that hosts our remote repository.

Remote repositories (GitHub)

- Hosts code (essentially the .git folder).
- Hosts issues / conversations / reference etc.
- is a git GUI.
- free account: All repos public (student exception)
- alternative: Bitbucket (offers free private repos)

Let's have a look: https://github.com/tidyverse/dplyr

GitHub

You can

- open issues.
- open pull request (ask the maintainer of a repo to pull your changes).
- merge branches
- etc.

=> Self study.

Remote repositories (GitHub)

Two scenarios:

- There is a repo somewhere and you want to *pull* it.
- You have a repo and want to *push* it somewhere.

Either way, go to GitHub first.

Remote repositories (pull)

- fork: Add a remote copy to your GitHub account (without downloading)
- clone your fork: “Download” your copy with `$git clone [url]`, e.g. `$git clone github.com/lorenzwalthert/styler`

Remote repositories (push)

Let's do it:

- Go to github.com.
- click "New Repository". Advised: use same name for repo as your local folder.
- Follow the instructions by c/p into terminal.
- reload github.com

=> Success?

Remote repositories (push)

- You just used `$git remote add [name of the remote] [url]`
- Your remote is usually called `origin`
- If there is a main repo that you forked before you cloned, it is usually called it `upstream`
- `git push --set-origin master` maps your `master` branch to the `master` in the remote branch and pushes the repo there.

Restoring previous versions

- Soft version: Revert (= undo *a* commit) = add a new commit on top of the tree.
- Hard version: Reset *to* a commit. Change your working directory to a state of the tree.

Soft version (= Revert)

- every commit has an id (= hash)
- see which commit has which hash with `$git log`
- use `$git revert [hash]`
- This will open your default text editor (probably VIM or VI by default).

Intermezzo: Using a text editor

- You can now edit the commit message.
- To enter the `insert` mode, press `i`
- You can now edit the commit message.
- When done, hit `ESC`
- Hit `:` (probably `Shift + .` on your keyboard)
- Then, type `wc` (for write and quit)
- Type `cq` to cancel and quit.
- Hit `ENTER` to confirm.

Soft version (= Revert)

Done. See your changes (how?)

Hard version (=Reset)

- every commit has an id (= hash).
- see which commit has which hash with `$git log`
- use `$git reset [hash] --hard` to reset a repo to that state.

Restoring previous versions

- You can also discard all changes in your working directory and return to the last commit you made:
- `$git reset HEAD --hard`
- In RStudio git tab. Right click on file -> revert.

Undo a `$git reset`

- Thank God you can.
- `$git log` shows commits in your tree.
- you can see all your actions (i.e. also commits you removed from your tree) with `$git reflog`.
- You removed the second commit from your tree before when you used `$git reset`. So how can you restore the state at the second commit?

Undo a `$git reset`

- find the hash with `$git reflog`.
- use `$git reset [hash] --hard`

More to explore (1)

May want to use Google / StackOverflow to anwser:

- `$git stash` / `$git stash pop` temporarily discard changes in working dir.
- `$git merge` / `$git rebase`.
- You can commit only parts of a file easily with the RStudio git tab without adding other parts to the index (and committing). How? Hint: Have a look at the diff.
- `$git branch` to switch between different branches.

More to explore (2)

- What is `HEAD`? What is `HEAD~1`? And what is `HEAD^1`? Hint: Used in contexts like this: `$git reset HEAD~1 --hard`.
- How to reset a single file instead of the whole tree?
- `$git rebase -i`. Interactive rebase (reorder commit, squash them together, drop them). Hint: try `$git rebase HEAD~2 -i` and have your text editor skills ready :-)
- `$git cherry-pick`. Add an arbitrary commit to your tree.
- `$git tag`. Anchor commits with tags.
- `$git diff` (between versions / between files). Hint: Configure a diff tool like merge diff.

More to explore (3)

- What does `$git commit --amend`
- How can you find all files tracked by git?
- What is `$git diff --cached`? Hint: Add a few files to the index and try `$git status` before.

A word of caution

- it's all in the .git/ folder. Be careful.
- no diff for binary files (e.g. word, pdf)
- use gitlfs (extension) for large files ($\sim > 1$MB).
- git is not a back-up system (unless you use a remote).

That's all

- Find my personal collection of commands here: http://bit.ly/ethz-tutorials-more-commands
- Contact me if you are stuck too long: lorwal@me.com
- Next week: R package develpment. Same time, same room.

Thanks for attending