Version Control – Version Management

Using **git**

Benjamin Hofner

benjamin.hofner@fau.de
Institut für Medizininformatik, Biometrie und Epidemiologie (IMBE)
Friedrich-Alexander-Universität Erlangen-Nürnberg

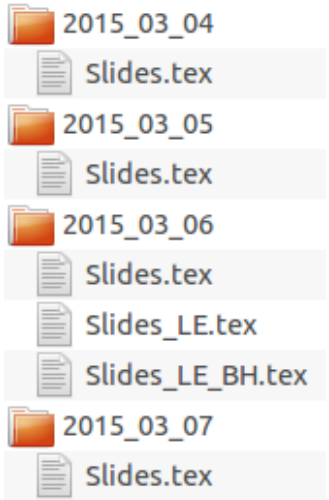
Biometrisches Kolloquium 2015

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
MEDIZINISCHE FAKULTÄT

Why Version Control?

Pros

- It helps you to *keep track of all changes*.
- It helps you to *cooperate* with others.
- It frees you of using files and folder such as given on the right ▶
- It stores your file history together with a change log.
- It helps you to perform internal (i.e., on your hard drive) and external (e.g., on the web or on a company server) *backups* of your files.
- You can always roll back to a specific version or day.
- It helps you to *review changes* ("difference viewer").



- ▶ All this helps to achieve **reproducible research** (as it helps you to reproduce your analysis etc.).

"Please find attached a zip folder containing the requested revised code. Thank you for providing the archive that I had submitted; **I experienced a system crash since submission and had lost several files**, *the archive with the code among them. I apologize for any inconvenience."*

Email from Author, 13.03.2015

What is the price to pay?

Cons

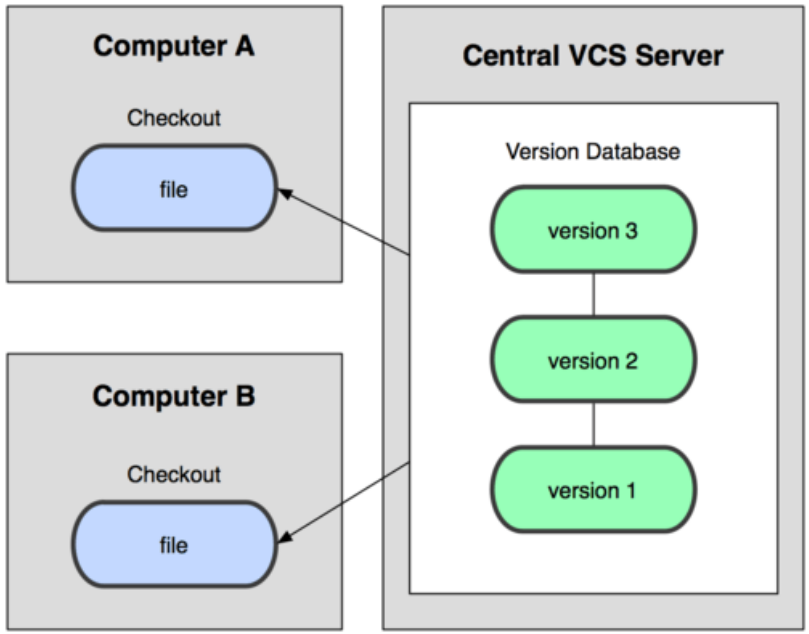
- You need to get accustomed to a new tool.
- You need to use this tool (actively), i.e., it requires some extra work to "commit" your files to the "repository".
- Git/SVN are not well suited for binary files such as Word files, figures, PDFs, ...

- **git** is very (!) powerful.

- If you want to use all/more features of **git** you need to
  a) use the shell and
  b) read the documentation and/or search the web.
- We cannot cover all details in this course. This is just a starter (but I will give you links to further good resources at the end of these slides).

Structure of Version Control Systems

- SVN

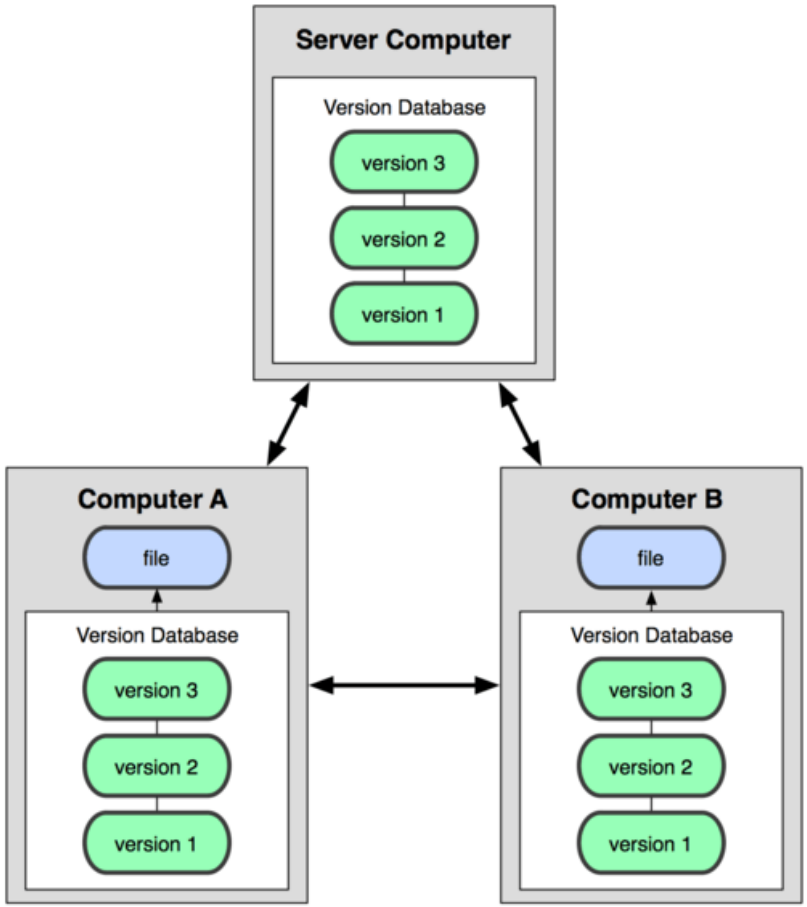


Source: http://git-scm.com/book/de/v1/
License: CC BY-NC-SA 3.0

- One central database that stores all changes.
- Users can `checkout` a version from the database and `commit` their changes to the database.

Structure of Version Control Systems

- **git**



Source: http://git-scm.com/book/de/v1/
License: CC BY-NC-SA 3.0

- On each computer we have a local database that stores all changes.
- Users can `checkout` a version from the (local) database and `commit` their changes to that database.
- Additionally, users can `clone` a database (e.g. from a remote server) and `push` updates to another database (e.g. to the remote server).

Important features of git

- Git can use a local repository in combination with remote repositories.
- Git allows us to use branches.
- One can amend a commit, if something is missing or was wrongly committed.
- Git can "speak" to SVN repositories (command `git svn ...`) and thus enriches SVN by the **git** features!

Using version control

Most important lesson to learn:
Commit early, commit often.

- Some people even state that "if the code isn't checked into source control, it doesn't exist." (and by code we can also think of other text-based files such as reports, papers, ...).
- Did I already tell you to **commit early and commit often**?

- No need to hesitate as one does not bother others with intermediate changes: They are local until you push them to the server!
- No need to hesitate as one can amend changes!
- If you want to experiment with your code: Create a branch and do it there. (see e.g., http://www.atlassian.com/git/tutorials)

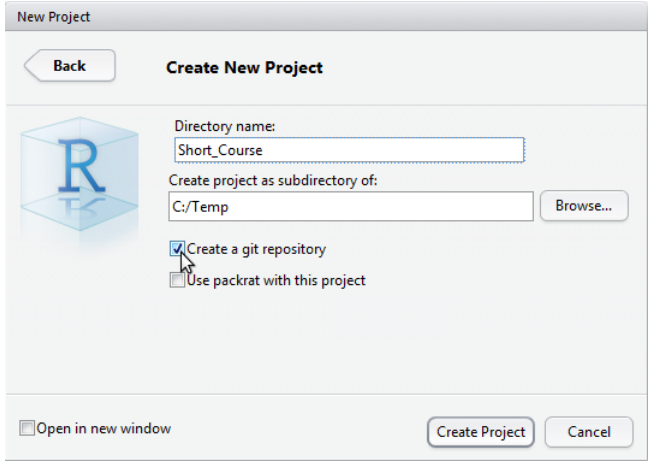
Let's go live

Setting up git with RStudio

- Install **git** from `http://git-scm.com/downloads`
- Add **git** to RStudio via
  `Tools` ▶ `Global options` ▶ `Git/SVN` ▶ `Git executable` ▶
  Enter path, e.g., `C:/Program Files (x86)/Git/bin/git.exe`

RStudio allows **git** support only for projects:

- Start new project
  `Tools` ▶ `Version Control` ▶ `Project Setup` ▶ `New Directory` ▶ `Empty Project` ▶ Choose directory and tick `Create a git repository`

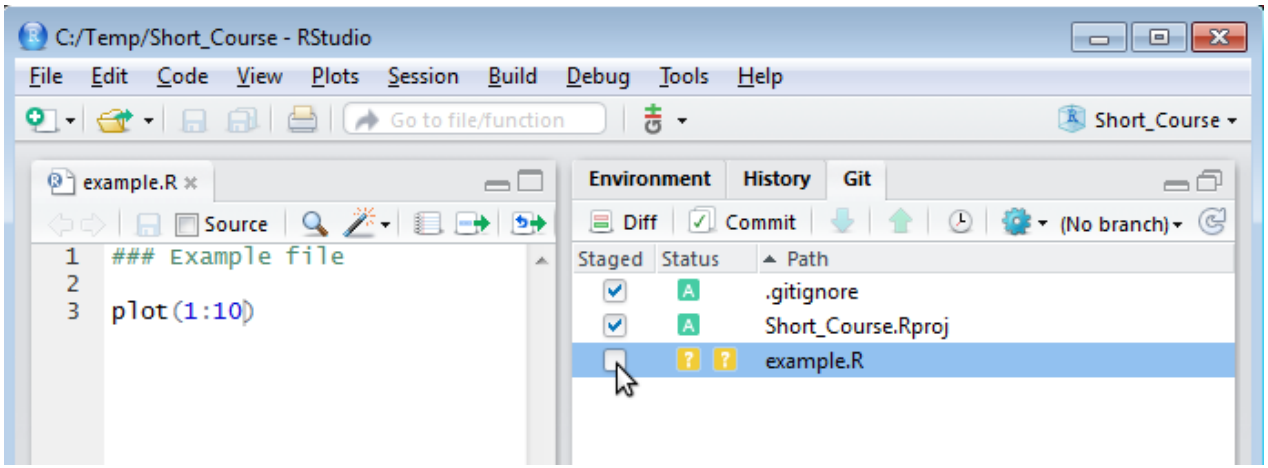


One could also import an exiting repository (see below) as a new project.

Let's go live

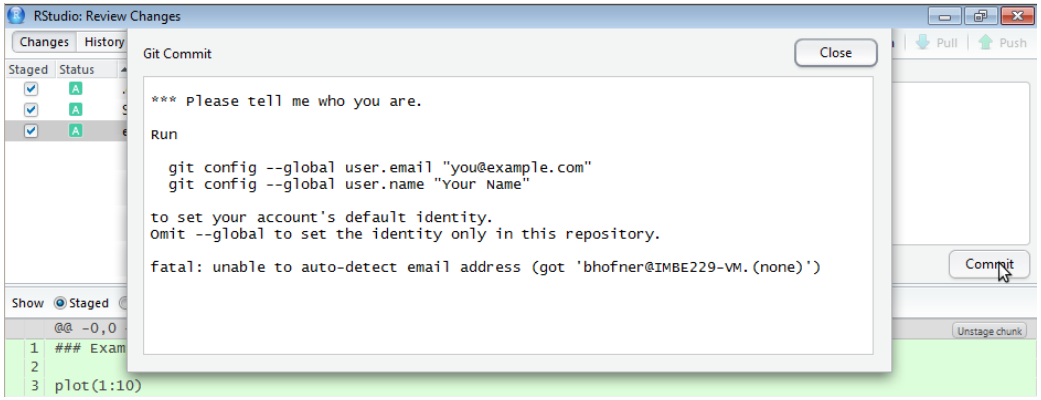
Staging and Committing

- Start creating files and save them in the project folder.
- Go to the git panel (push `STRG-9`)
- Start "staging" files (i.e., adding them to version control)
- (Only) staged files are included in the next `commit` (see next step)

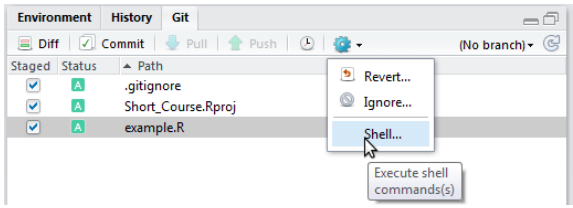


- `Commit` the changes to the (local) database
- Enter a descriptive log message:
  - Summary in the first line
  - (Details in the following lines)

- ERROR as we did not initialize **git**:



  - Start `Shell` (this is always required if non-standard commands are to be used)



- enter required commands.
  - Check results with

```
git config --global -l
```

More details and help: R Studio Support Page

Go Online

Using GitHub

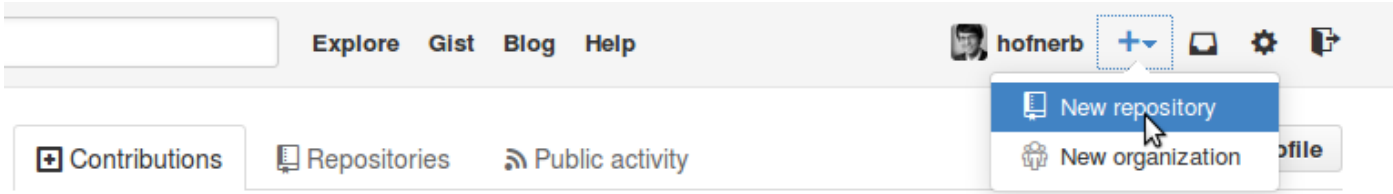
- Create an user account on GitHub (`http://github.com/join`).
- Create an SSH key for authentication (password no longer needed):
  - Go to `Tools` ▶ `Global Options ...` ▶ `Git/SVN` ▶ `Create RSA key...`
  - If this fails create the missing folder, e.g., `C:/Users/Benjamin/.ssh` (in Windows Explorer create a new folder in your user folder and name it `.ssh.` (with a dot in the beginning and end)).
  - Retry.
  - If it still fails type in the shell:

```
ssh-keygen -t rsa -C "your_email@address.com"
```

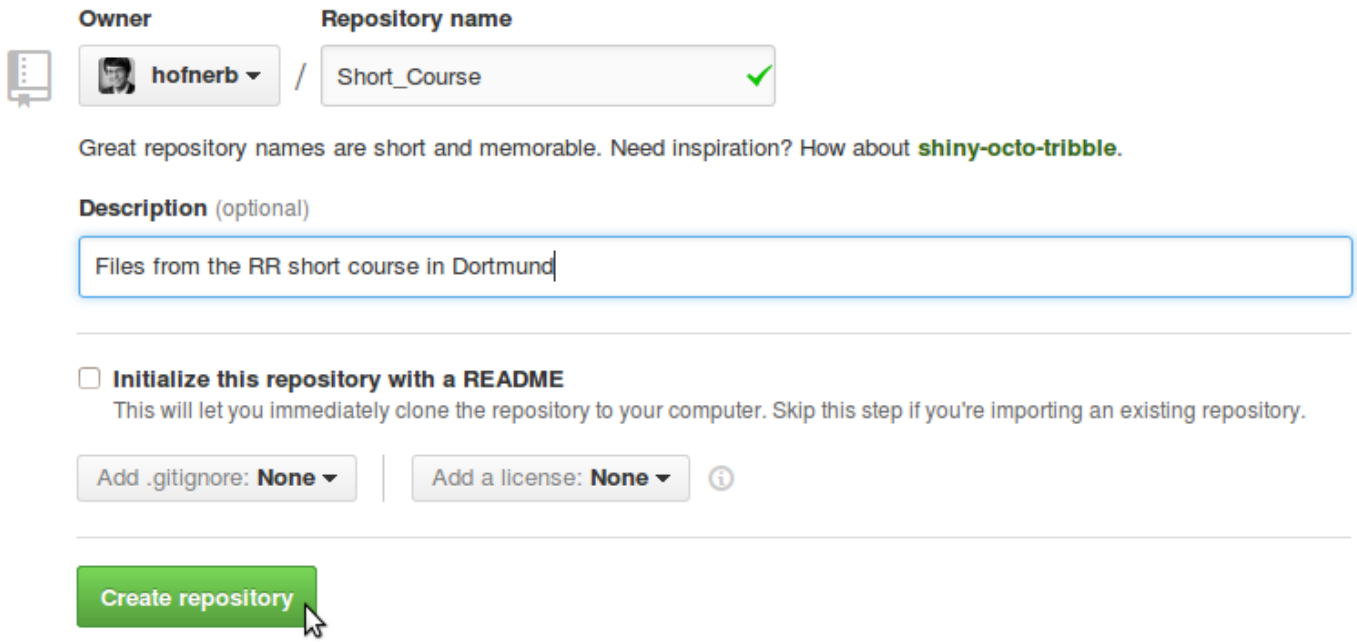
- Go to `Tools` ▶ `Global Options ...` ▶ `Git/SVN` ▶ `View public key`
- Copy the key to `https://github.com/settings/ssh`
- Now we are able to set up a remote server for our git repository.

Set up a new repository

- On your GitHub page click "+" and "New repository" or go to `https://github.com/new`

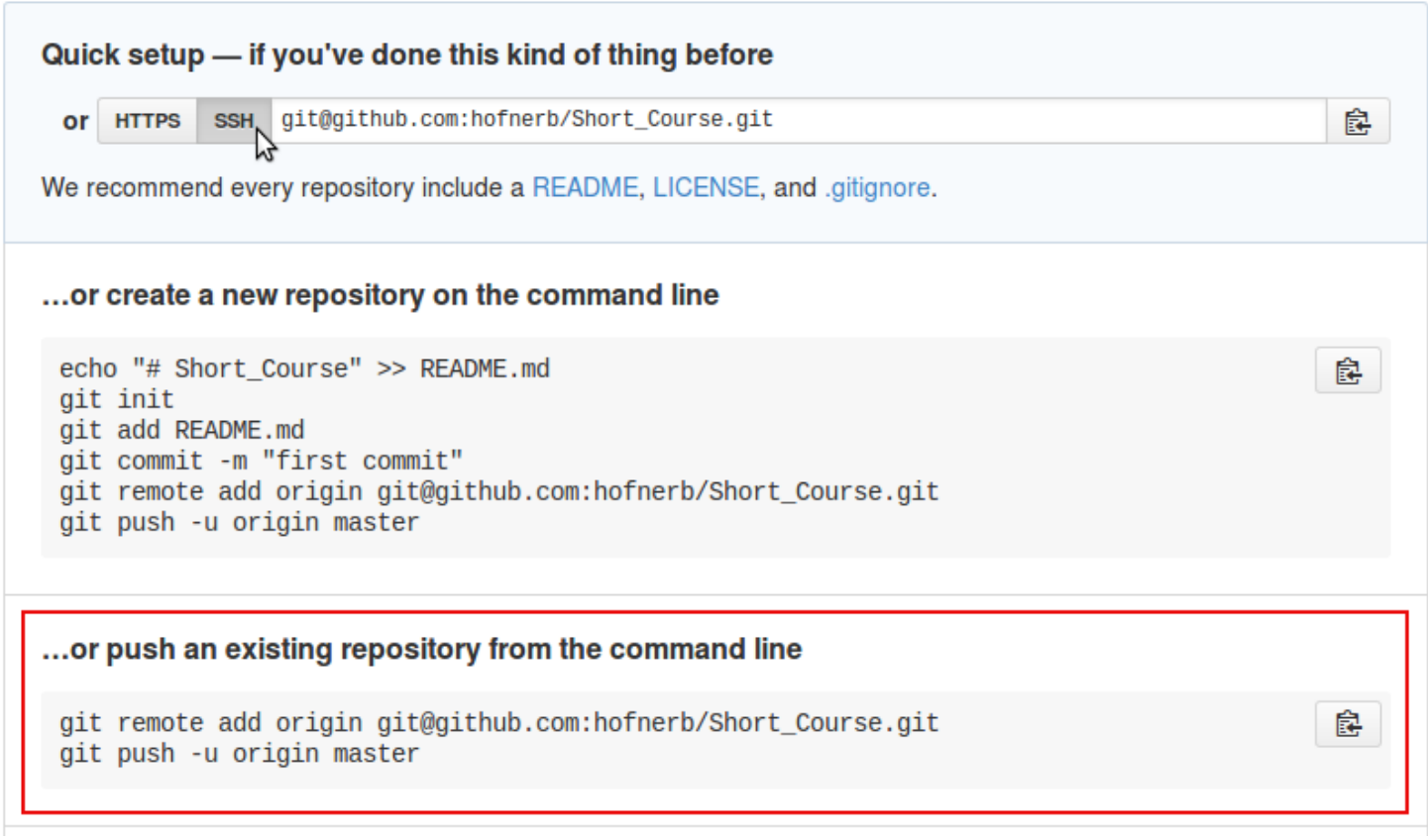


- Name your new repository and click "Create repository"



Set up a new repository

- Select `SSH`
- Copy the highlighted commands to your shell.



- *Done!*

- Now we can start `pushing` our `commits` to the remote repository.

What's next?

From now on,

- we make changes,
- `add` the changes (or new files) to the staging area,
- `commit` them to the local database,
- and `push` these changes to the remote repository from time to time.

If we work in teams or use multiple machines we need another step,

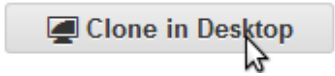
- we need to `pull` changes from the remote repository to our local repository (this `fetch`es the changes from the remote to the local database and `merge`s them with the local version)

At any time,

- we can inspect `diff`erences of the local version with the committed version,
- we can `revert` local changes,
- ...

To access all features of **git** one needs to learn more (see resources) and then use the shell directly (i.e., type your commands).

Other clients

- One can use for example other graphical user interfaces (GUIs) such as `gitk` or `git gui`.
- For Windows and Mac, GitHub also offers a special GUI. In your project simply click the button 

and follow the installation process.

For more details see `http://git-scm.com/book/en/v2/Git-in-Other-Environments-Graphical-Interfaces`

Outlook: Controlling Package Versions

- Until now, we have learned to keep track of changes in our code and other files.
- *But what if R or R packages change?*

There exist some solutions that facilitate the management of installed R packages.

- **checkpoint**: Install packages from snapshots on the checkpoint server for reproducibility.
  (Snapshot archives exist starting from 2014-09-17.)
  URL: http://cran.r-project.org/package=checkpoint
  - Documentation

```
> library("checkpoint")
> checkpoint("YYYY-MM-DD")
```

- **packrat**: A dependency management system for projects and their R package dependencies.
  URL: http://cran.r-project.org/package=packrat
  - Documentation

  (included in RStudio)

Odds and ends

What if ...

- the R version changes?
- the hardware changes? (e.g. 32-bit / 64-bit)?
- the operating system (OS) changes? (Windows, Linux, Mac OS)?
- ...

- R packages might be a solution to some of these problems (especially if they are checked daily on CRAN) but not a common solution to all these problems in all circumstances.
- Virtual machines might also help to create fully reproducible environments.

Summary

- Tools such as **git** ease the reproducibility of analysis etc.
- However, it requires some extra work,
- but if one gets used to it, it usually is very quick (and much quicker than to recreate lost code).
- No common solution to all requirements exists.

Further Resources

- Get free **private** GitHub repositories:
  https://education.github.com/
  - List of other git repositories:
    https://git.wiki.kernel.org/index.php/GitHosting

For advanced users:

- Git book:
  EN: http://git-scm.com/book/en (Vers 2)
  DE: http://git-scm.com/book/de (Vers 1)
  - Git for subversion users:
    http://git.or.cz/course/svn.html
  - Using `git svn`:
    http://git-scm.com/book/en/v2/Git-and-Other-Systems-Git-as-a-Client
  - More on `git svn`:
    http://trac.parrot.org/parrot/wiki/git-svn-tutorial

Finally, a note of caution:

- (Large) binary files should not be used in the main repository as it slows down everything: ▶ Use a separate repository and `git submodule` (details)