Reproducible research with R

12th June 2013

Outline

Part I – version control

- Introduction
  - Reproducibility of analysis
  - Version control
- Project design
  - What is version controlled?
  - Data persistence
- Using version control with R
  - Setting up a project
  - Using Git with R

Outline

Part I – version control

Outline

Part I – version control

- Introduction
  - Reproducibility of analysis
  - Version control
- Project design
  - What is version controlled?
  - Data persistence
- Using version control with R
  - Setting up a project
  - Using Git with R

Outline

Part I – version control

- Introduction
  - Reproducibility of analysis
  - Version control
- Project design
  - What is version controlled?
  - Data persistence
- Using version control with R
  - Setting up a project
  - Using Git with R

Outline

Part II – Report generation

- Introduction
  - Report generation
  - Tools
- Markdown – HTML output
  - Markdown language
  - Exercises with R
- Sweave and LateX – PDF output
  - Latex
  - Exercises with R

Outline

Part II – Report generation

- Introduction
  - Report generation
  - Tools
- Markdown – HTML output
  - Markdown language
  - Exercises with R
- Sweave and LateX – PDF output
  - Latex
  - Exercises with R

Outline

Part II – Report generation

- Introduction
  - Report generation
  - Tools
- Markdown – HTML output
  - Markdown language
  - Exercises with R
- Sweave and LateX – PDF output
  - Latex
  - Exercises with R

Outline

Part II – Report generation

- Introduction
  - Report generation
  - Tools
- Markdown – HTML output
  - Markdown language
  - Exercises with R
- Sweave and LateX – PDF output
  - Latex
  - Exercises with R

Outline

Bonus – figure generation with `make`

- Introduction
  - Presentation of `make`
  - Interest for figure building
- Using `make`
  - Rules
  - Simple example
- Automatic figure building
  - Exercises with R and `make`

Outline

Bonus – figure generation with `make`

- Introduction
  - Presentation of `make`
  - Interest for figure building
- Using `make`
  - Rules
  - Simple example
- Automatic figure building
  - Exercises with R and `make`

Outline

Bonus – figure generation with `make`

- Introduction
  - Presentation of `make`
  - Interest for figure building
- Using `make`
  - Rules
  - Simple example
- Automatic figure building
  - Exercises with R and `make`

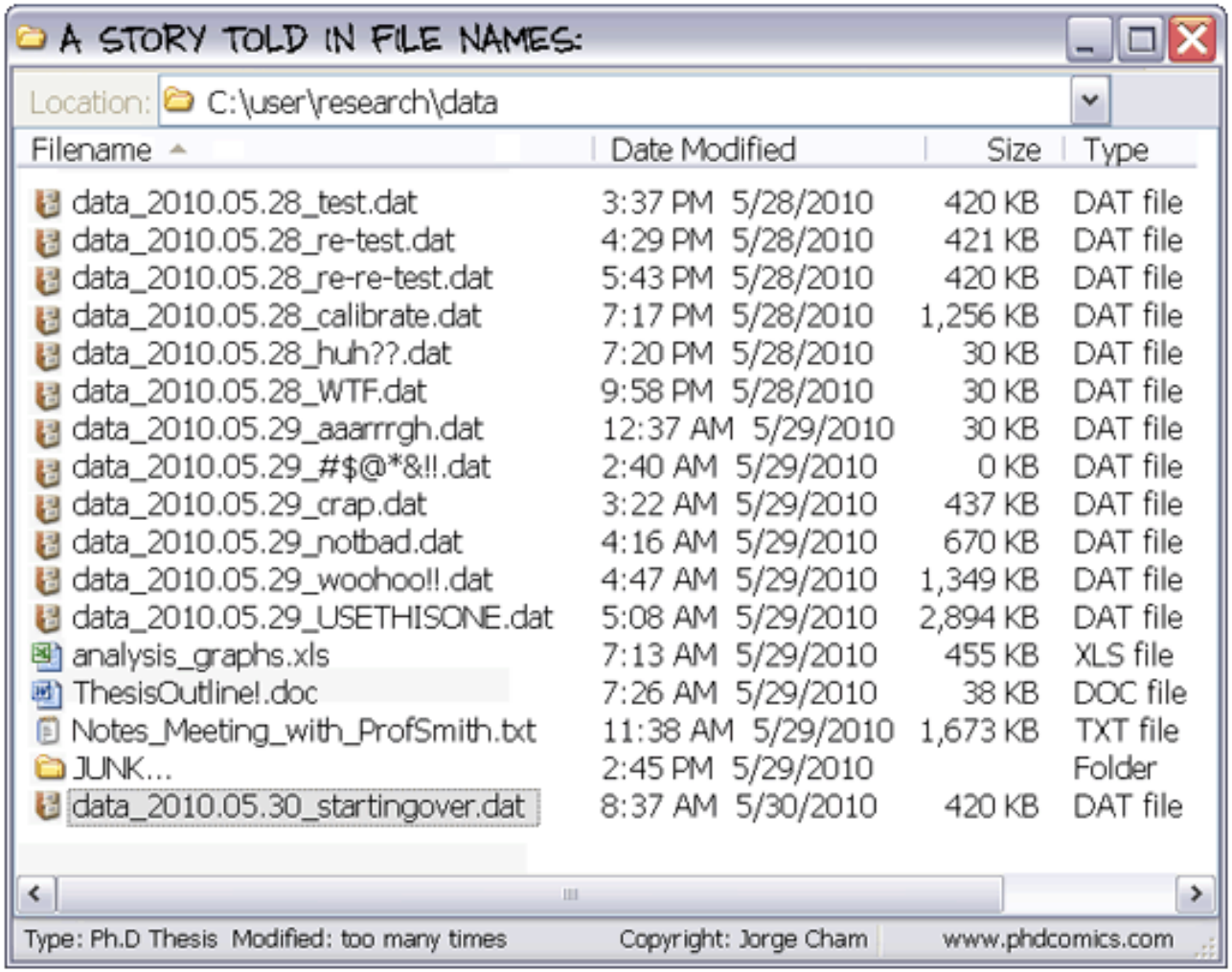
Outline

Bonus – figure generation with `make`

- Introduction
  - Presentation of `make`
  - Interest for figure building
- Using `make`
  - Rules
  - Simple example
- Automatic figure building
  - Exercises with R and `make`

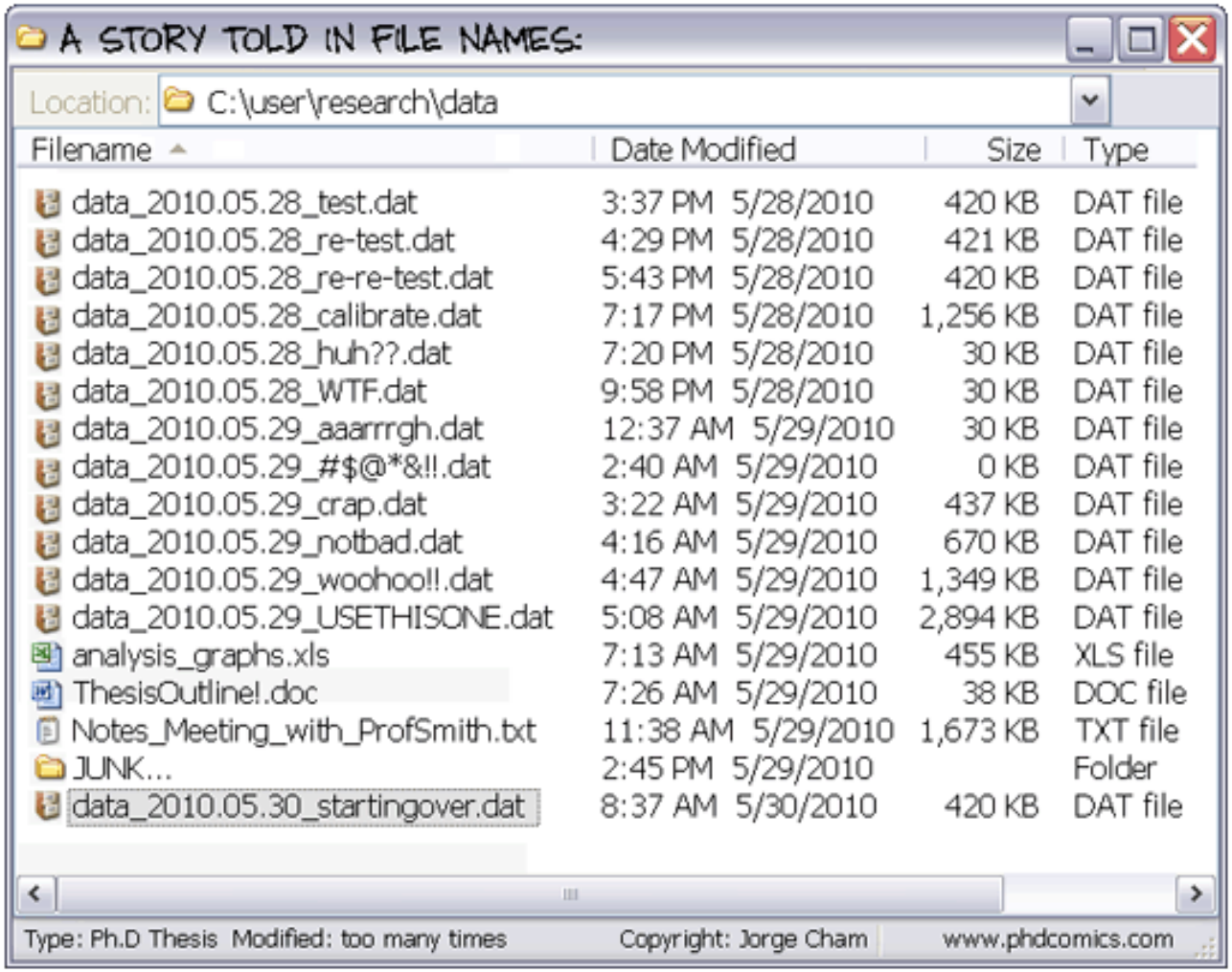
Part I
General concepts
Version control

A common problem



Not only for data, but for scripts also

A common problem



Not only for data, but for scripts also

Reproducibility?

- Research should be reproducible by others.
- This refers to the experiments generating the data, but also to the analysis of the data.
- The first researcher who will need to reproduce the results is likely to be you.

Reproducibility?

- Research should be reproducible by others.
- This refers to the experiments generating the data, but also to the analysis of the data.
- The first researcher who will need to reproduce the results is likely to be you.

Reproducibility?

- Research should be reproducible by others.
- This refers to the experiments generating the data, but also to the analysis of the data.
- The first researcher who will need to reproduce the results is likely to be you.

Reproducibility of analysis

- Lab books make lab work traceable. Analysis should also be traceable.
- Analysis steps must be recorded, and reverting to any previous step must be possible.
- This ensures that we always exactly know how a result was generated.

Reproducibility of analysis

- Lab books make lab work traceable. Analysis should also be traceable.
- Analysis steps must be recorded, and reverting to any previous step must be possible.
- This ensures that we always exactly know how a result was generated.

Reproducibility of analysis

- Lab books make lab work traceable. Analysis should also be traceable.
- Analysis steps must be recorded, and reverting to any previous step must be possible.
- This ensures that we always exactly know how a result was generated.

Version control

- Version control is a tool to keep track of file changes.
- However, version control softwares offer more than simply recording successive versions of a file.
- Version controlled projects can be splitted, merged and shared with collaborators.

Version control

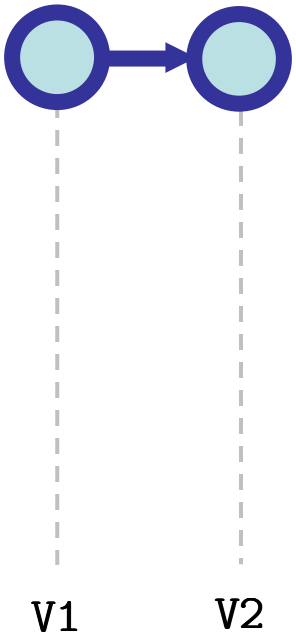
- Version control is a tool to keep track of file changes.
- However, version control softwares offer more than simply recording successive versions of a file.
- Version controlled projects can be splitted, merged and shared with collaborators.

Version control

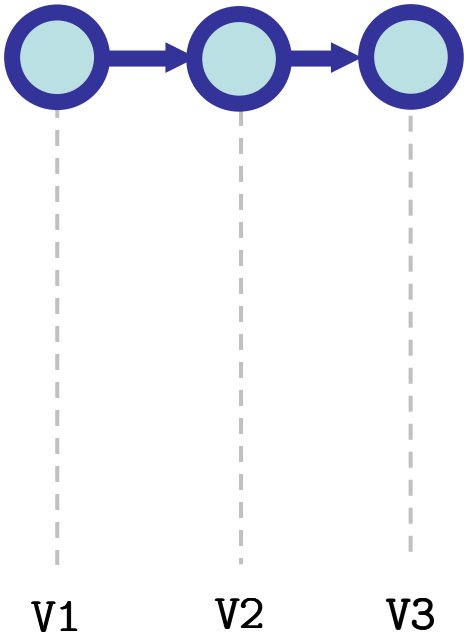
- Version control is a tool to keep track of file changes.
- However, version control softwares offer more than simply recording successive versions of a file.
- Version controlled projects can be splitted, merged and shared with collaborators.

Version control flow

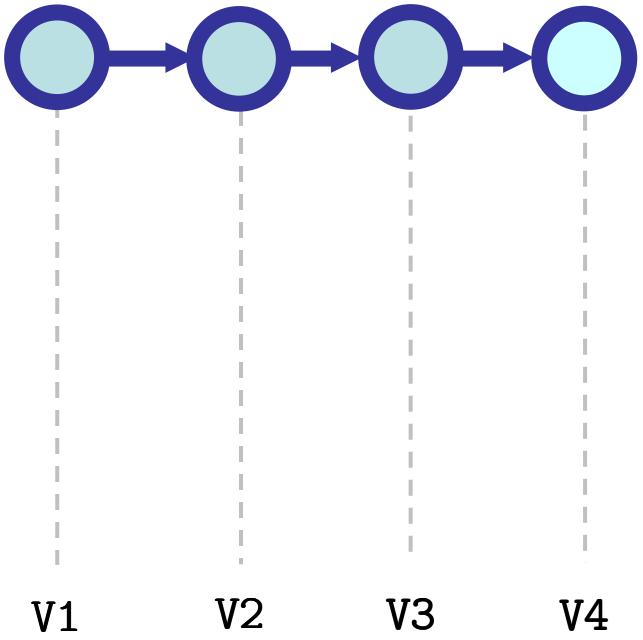
Version control flow



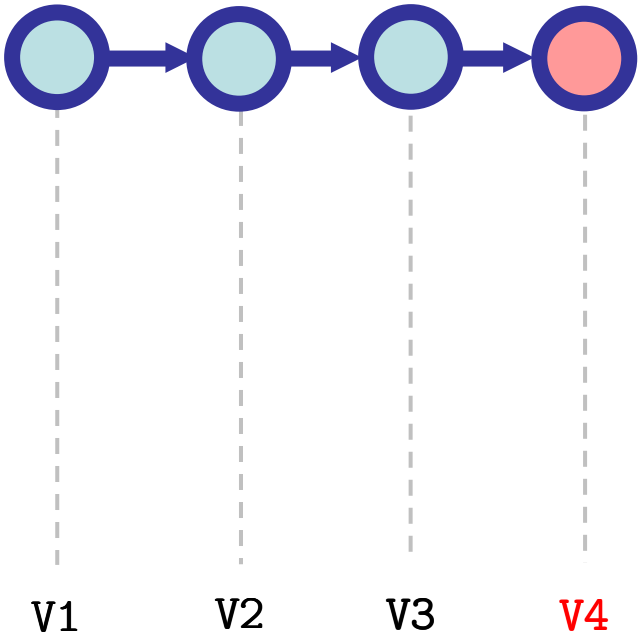
Version control flow



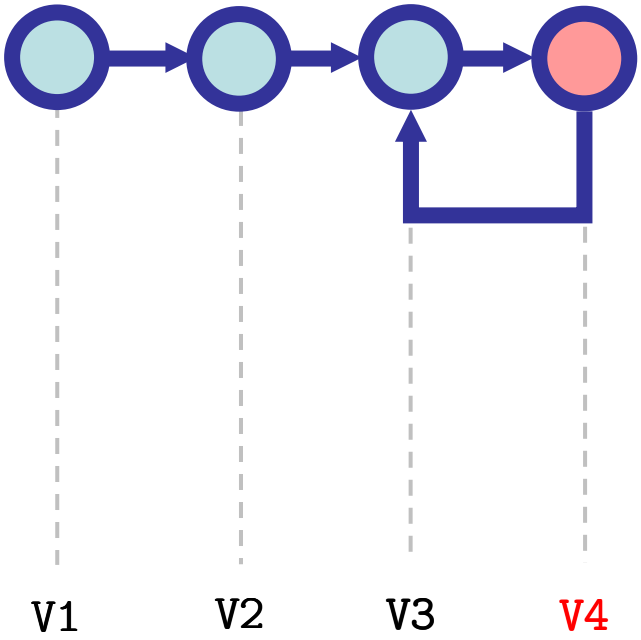
Version control flow



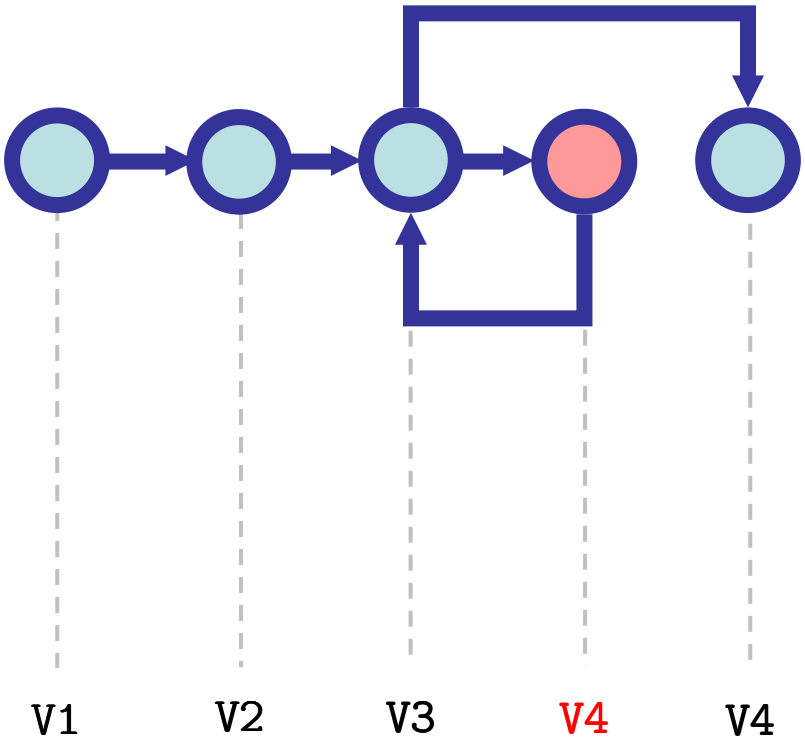
Version control flow



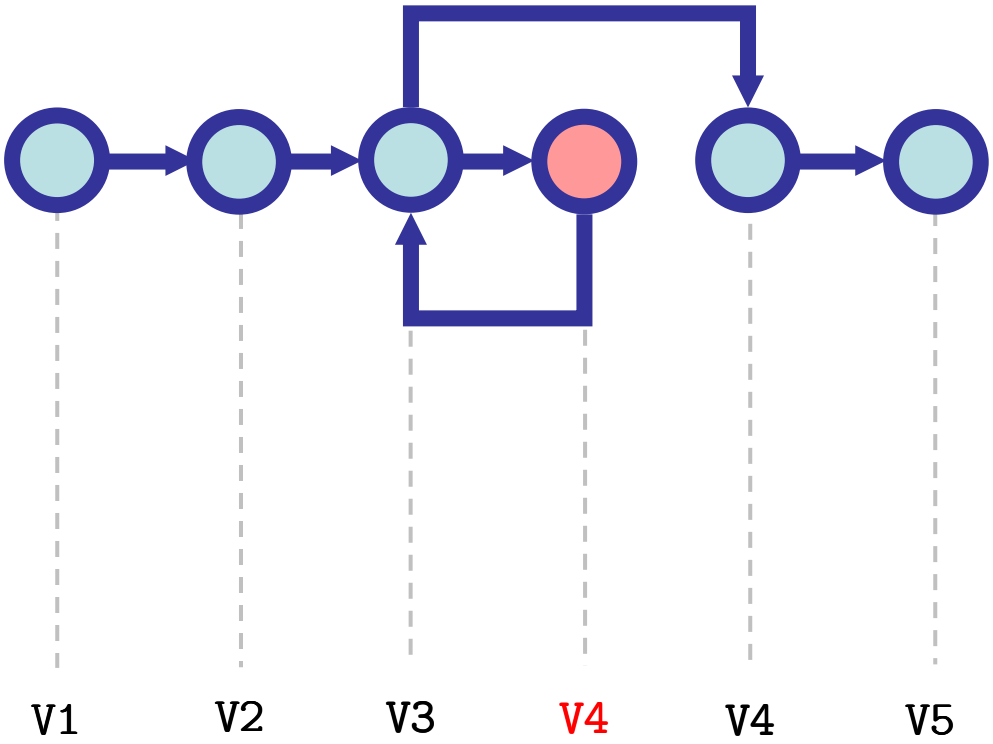
Version control flow



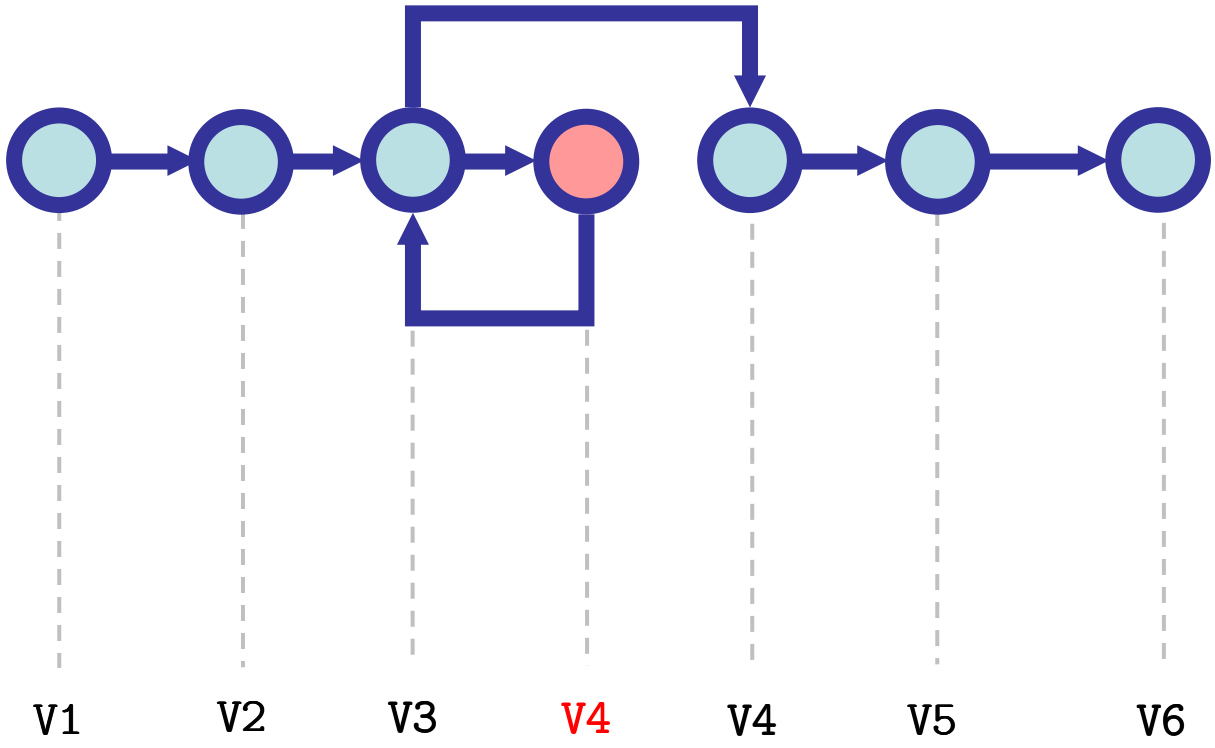
Version control flow



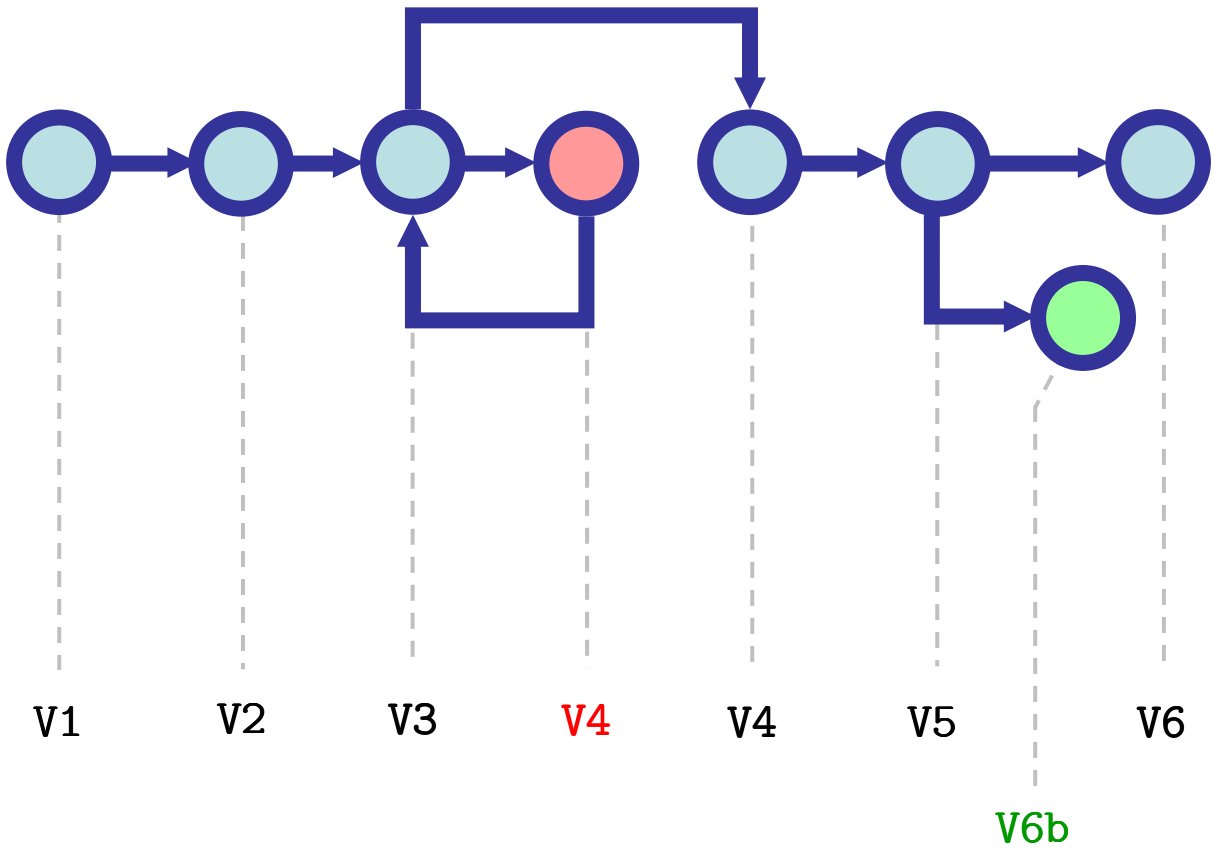
Version control flow



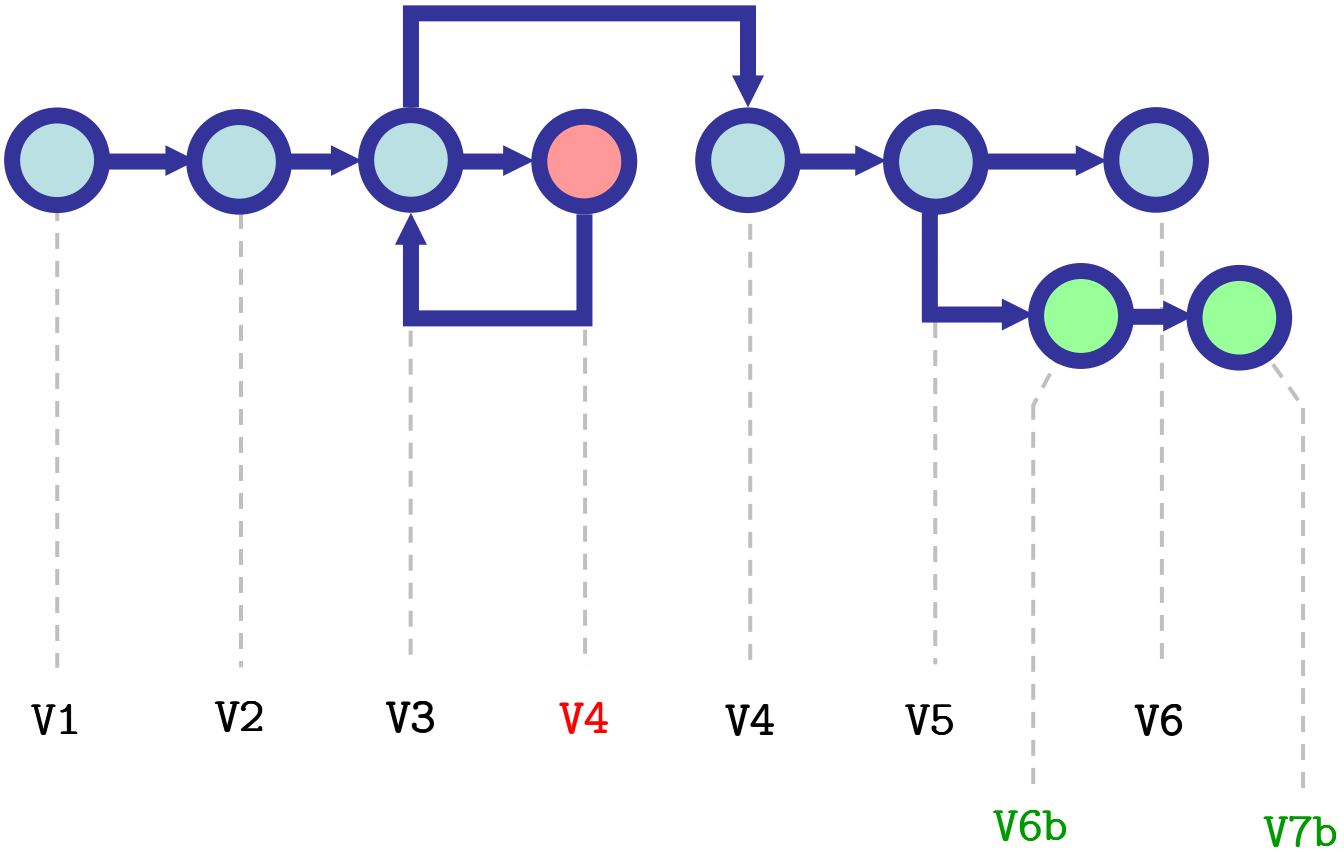
Version control flow



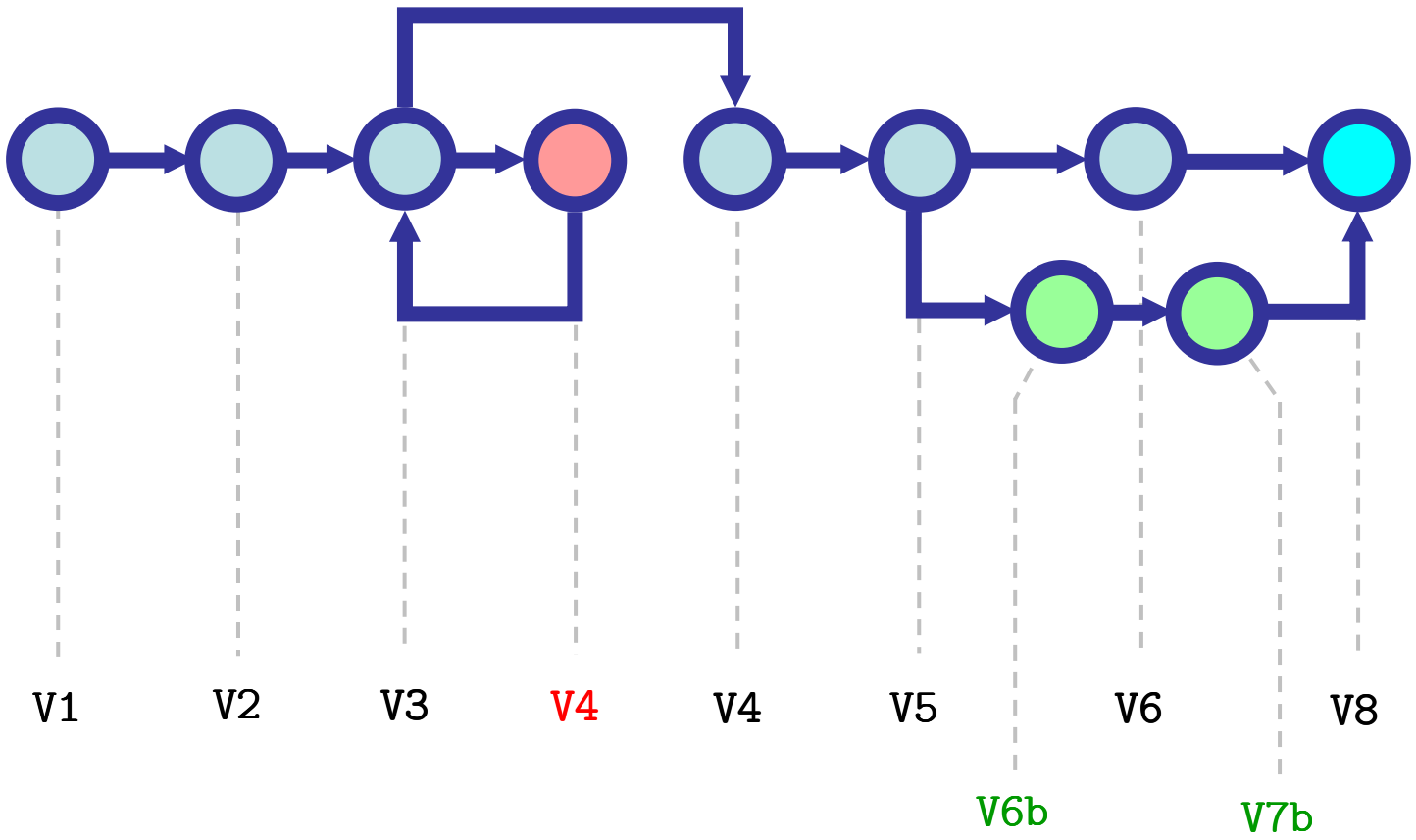
Version control flow



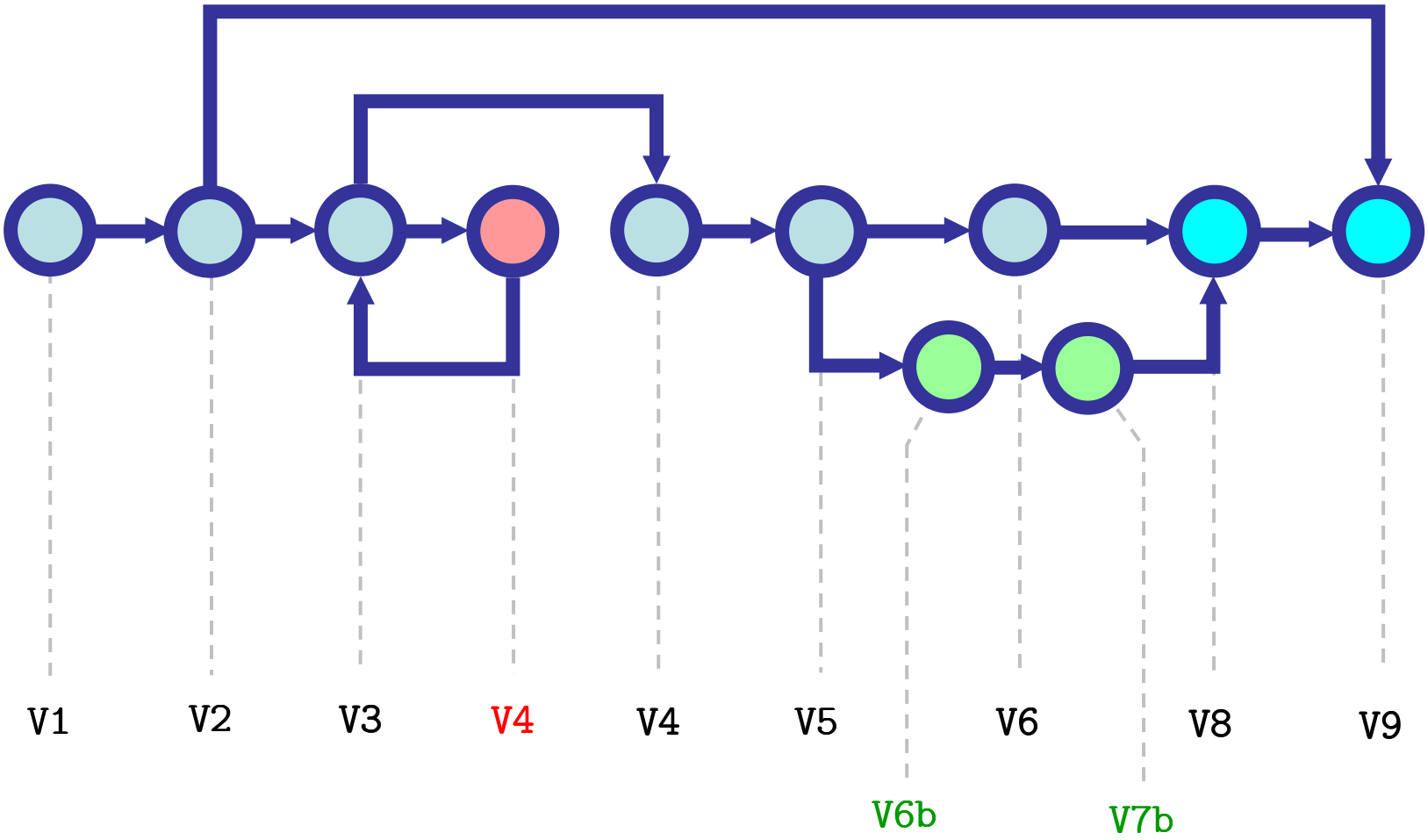
Version control flow



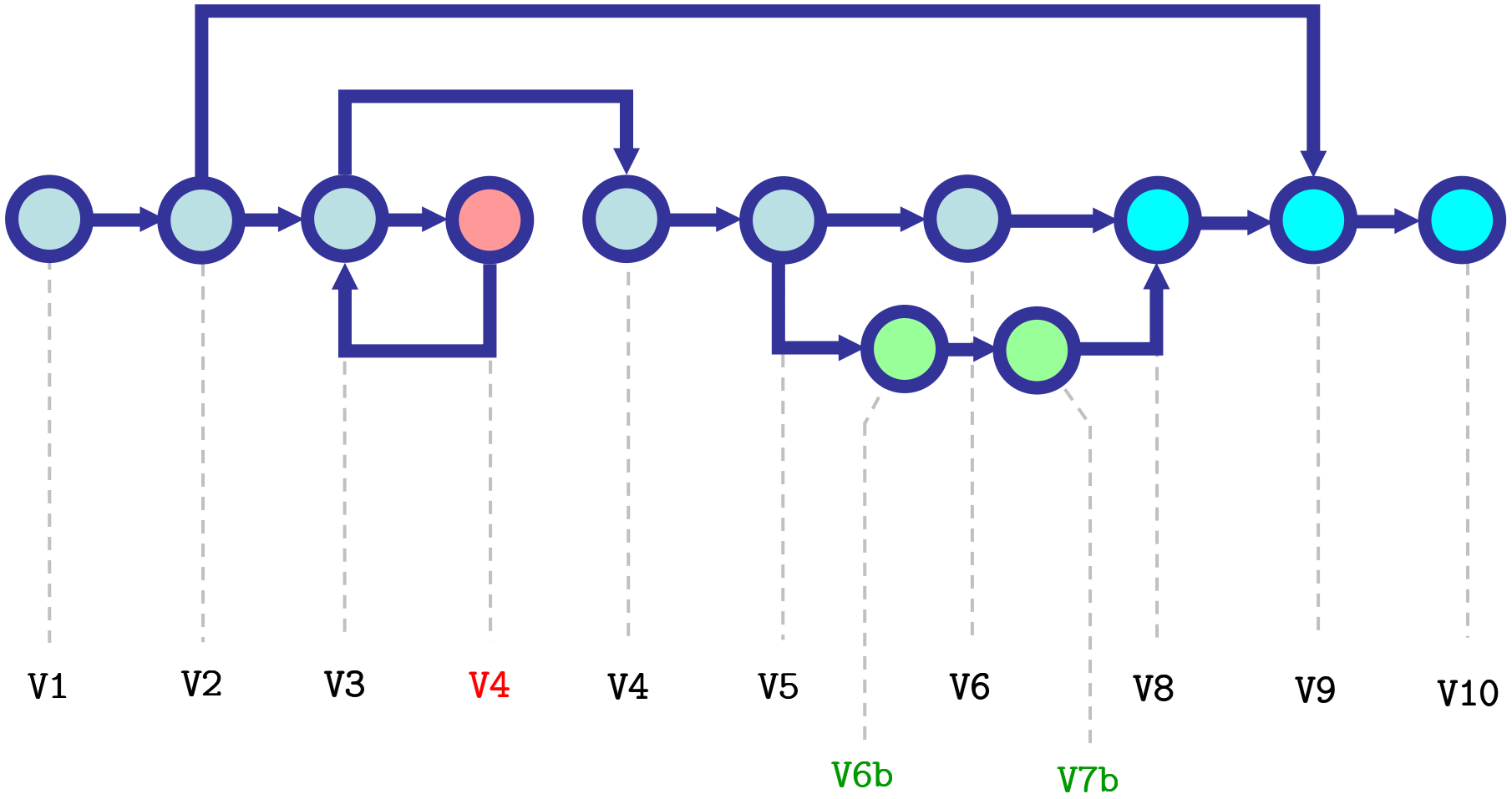
Version control flow



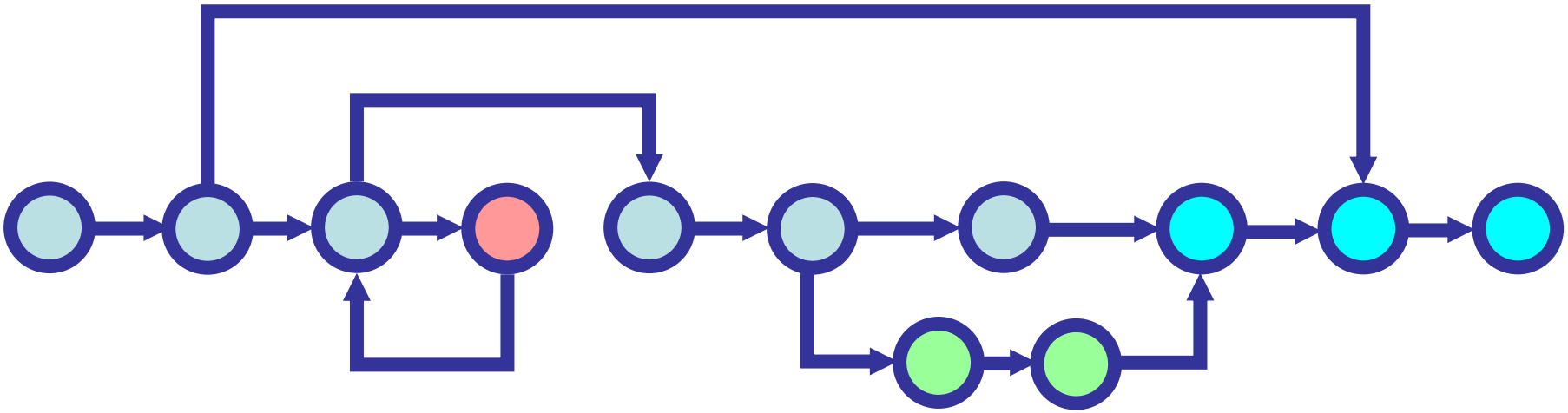
Version control flow



Version control flow

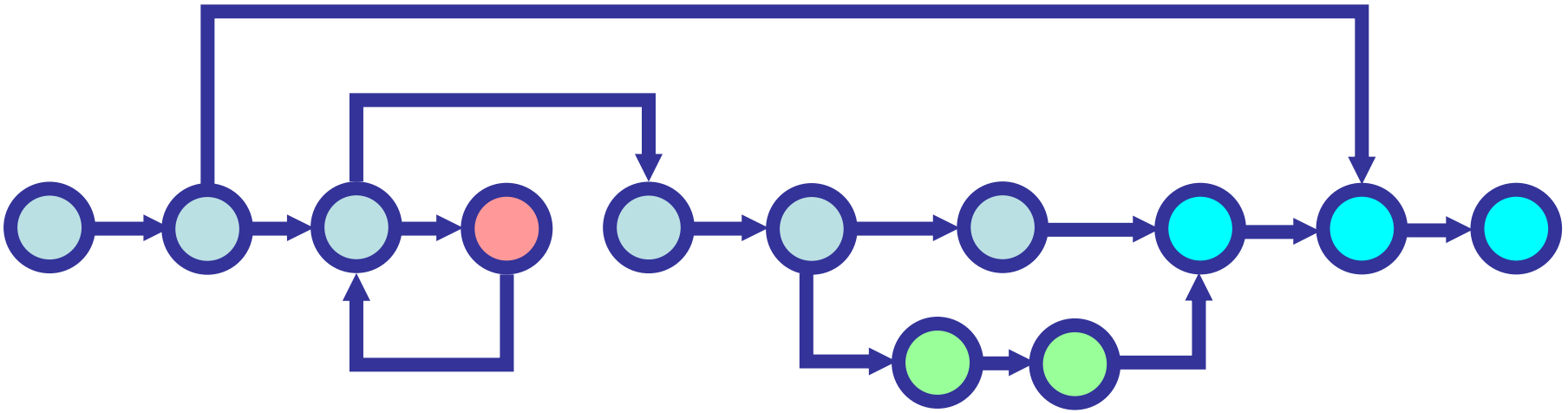


Version control flow



- In the usual approach, each version would have been a set of different files, with not-so-meaningful names.
- With version control, every change is tracked, can be reverted, and each branch is a clean set of files.

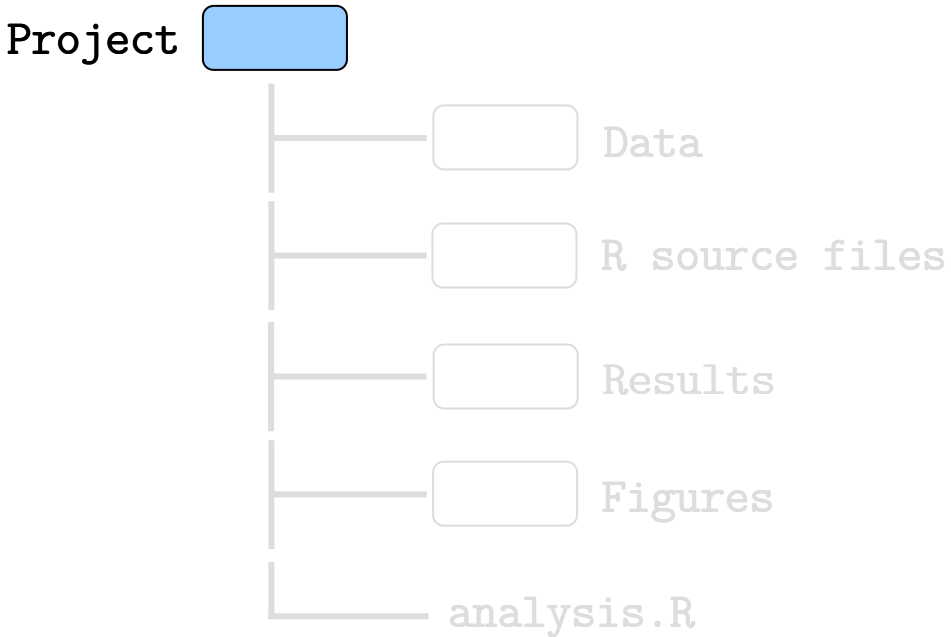
Version control flow



- In the usual approach, each version would have been a set of different files, with not-so-meaningful names.
- With version control, every change is tracked, can be reverted, and each branch is a clean set of files.

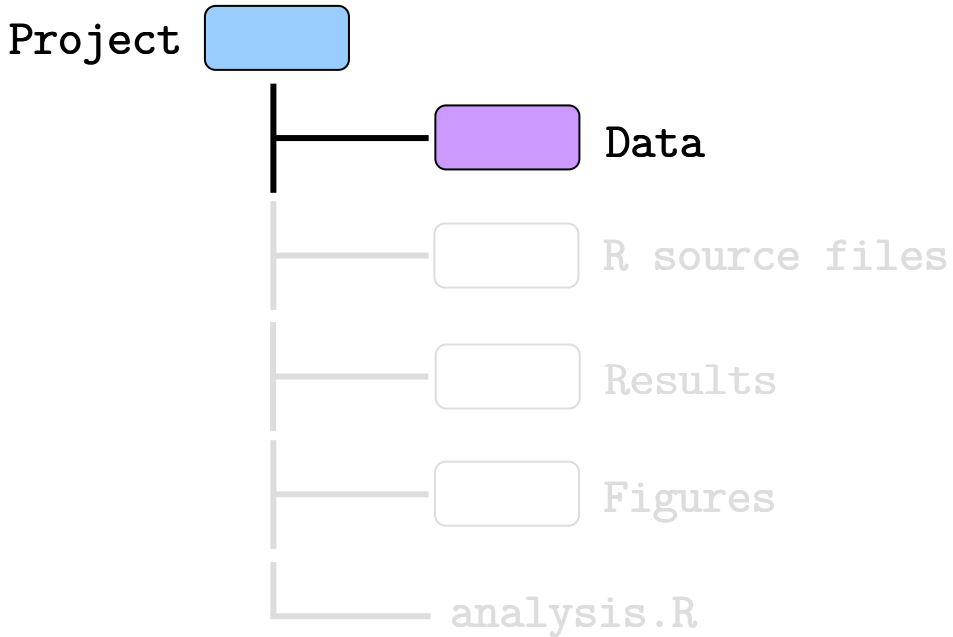
Project design

Structure of a project folder



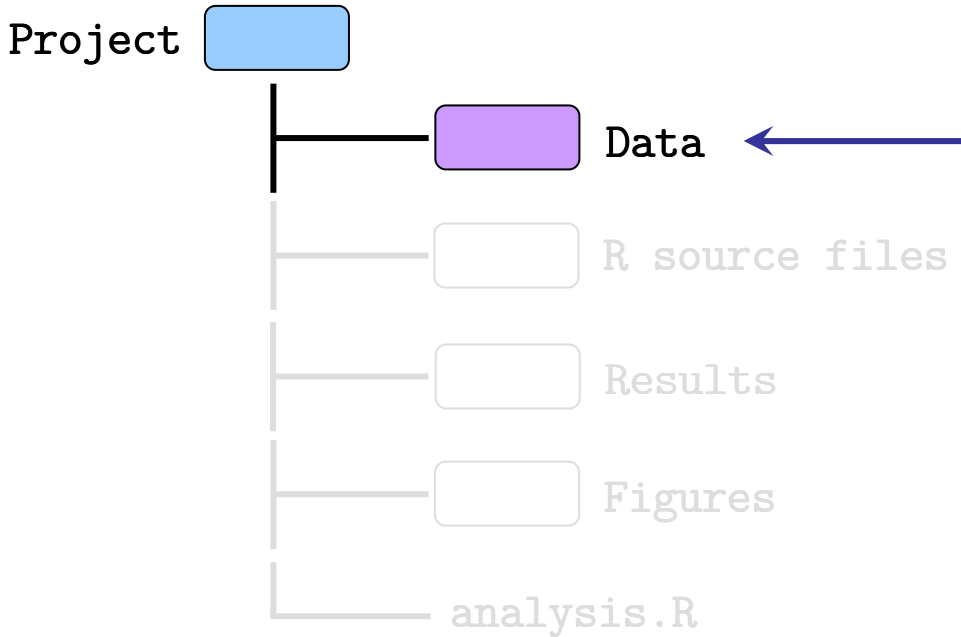
(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

Structure of a project folder



(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

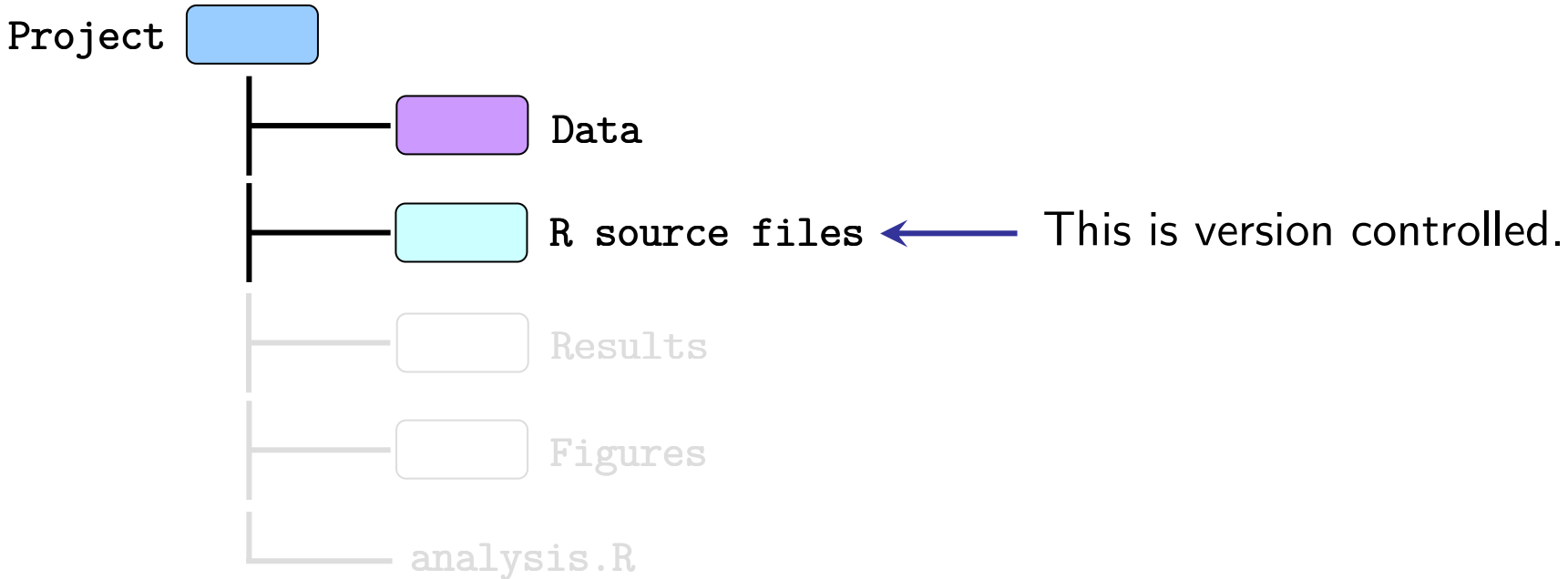
Structure of a project folder



This is read-only. It is not version controlled but it is backed up.

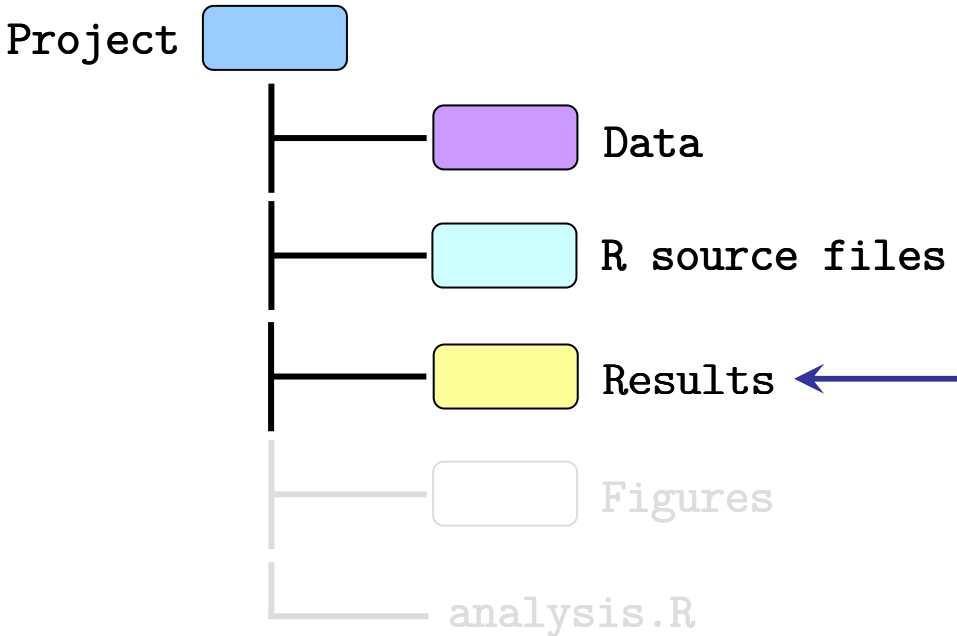
(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

Structure of a project folder



(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

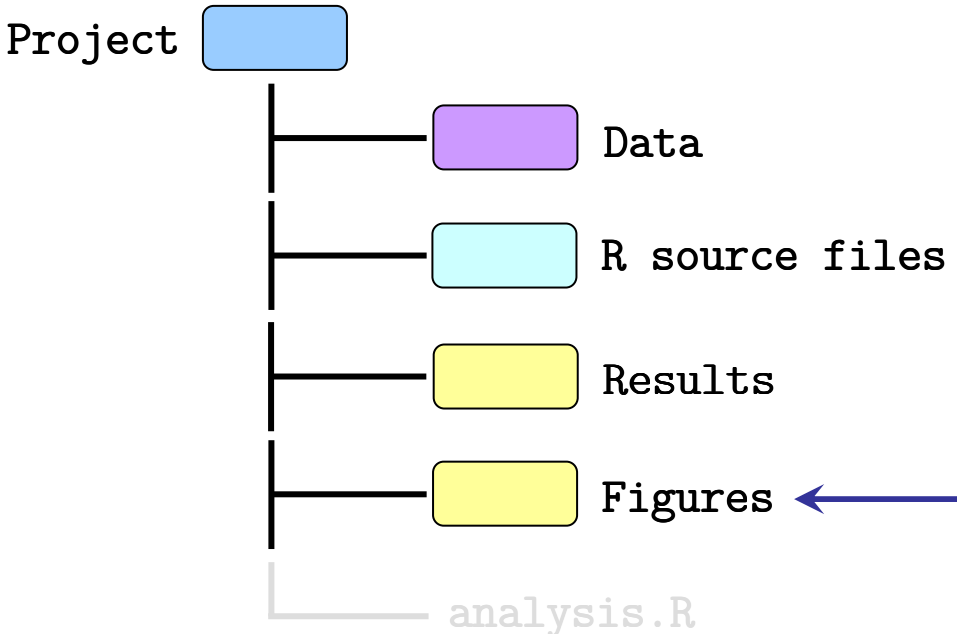
Structure of a project folder



This is generated from the two previous. It is not version controlled nor backed up. It is disposable.

(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

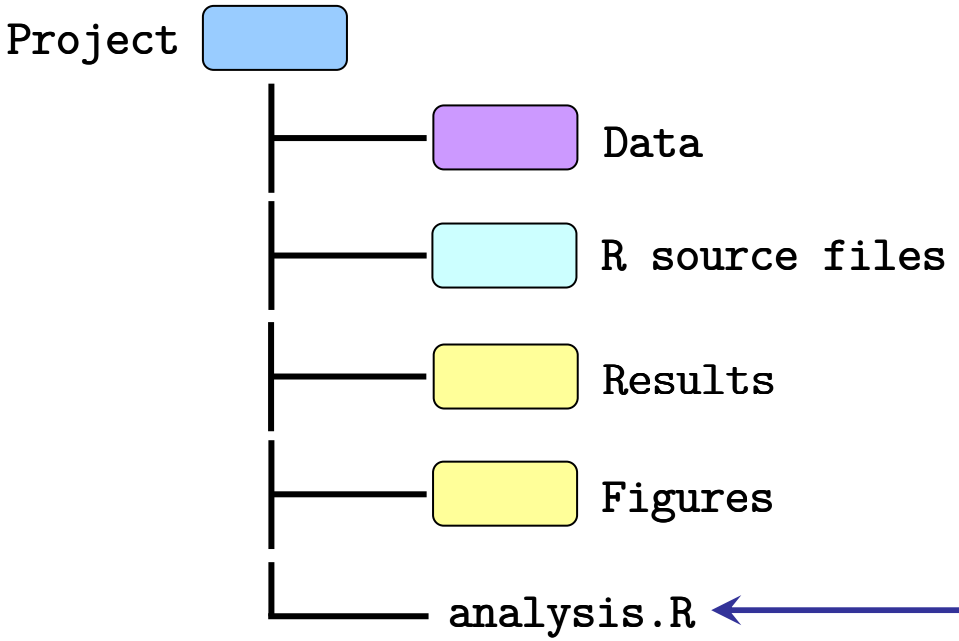
Structure of a project folder



This is also generated from the two previous and disposable.

(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

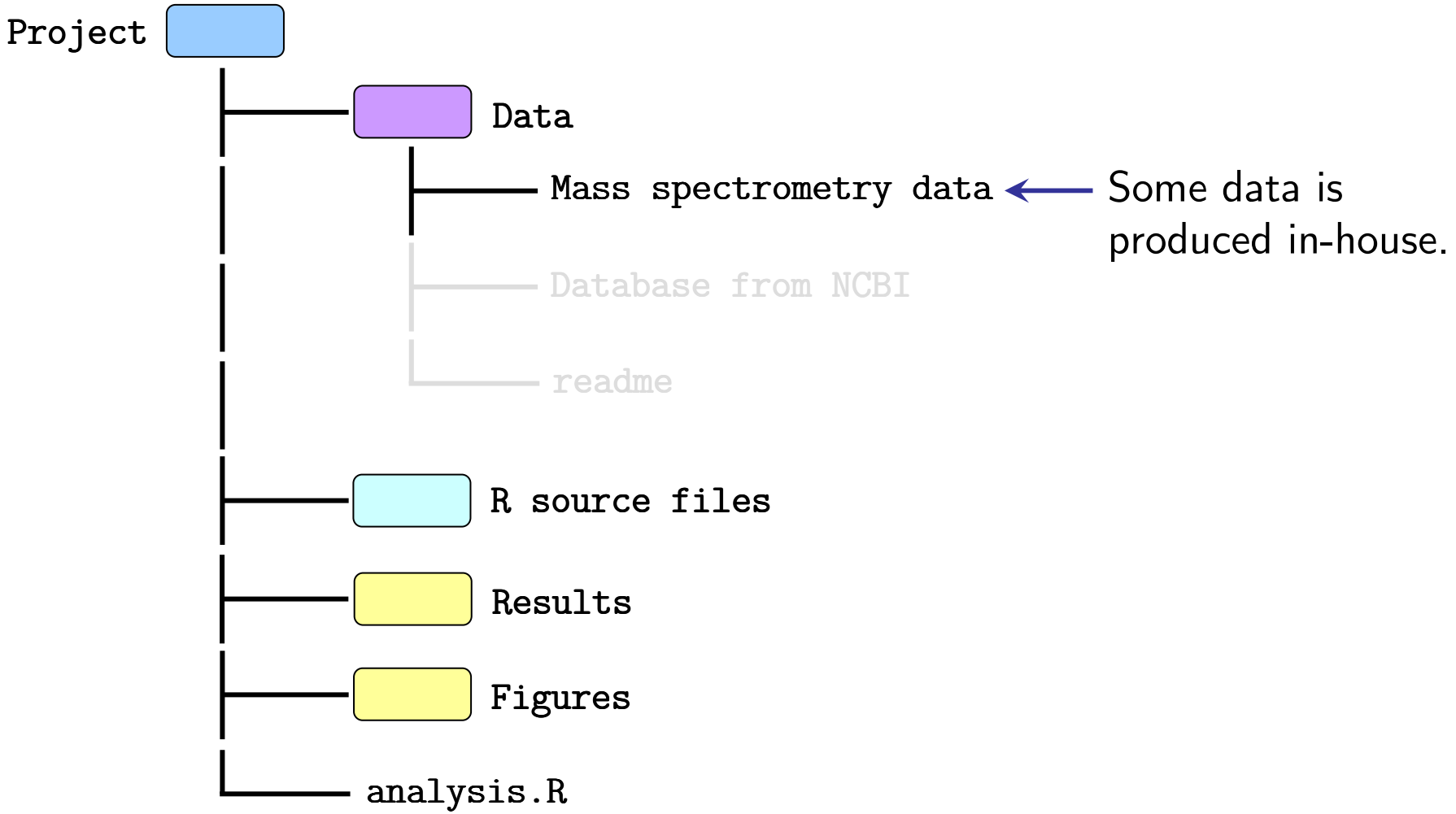
Structure of a project folder



This is the main script using the R source files to do the analysis. It is version controlled.

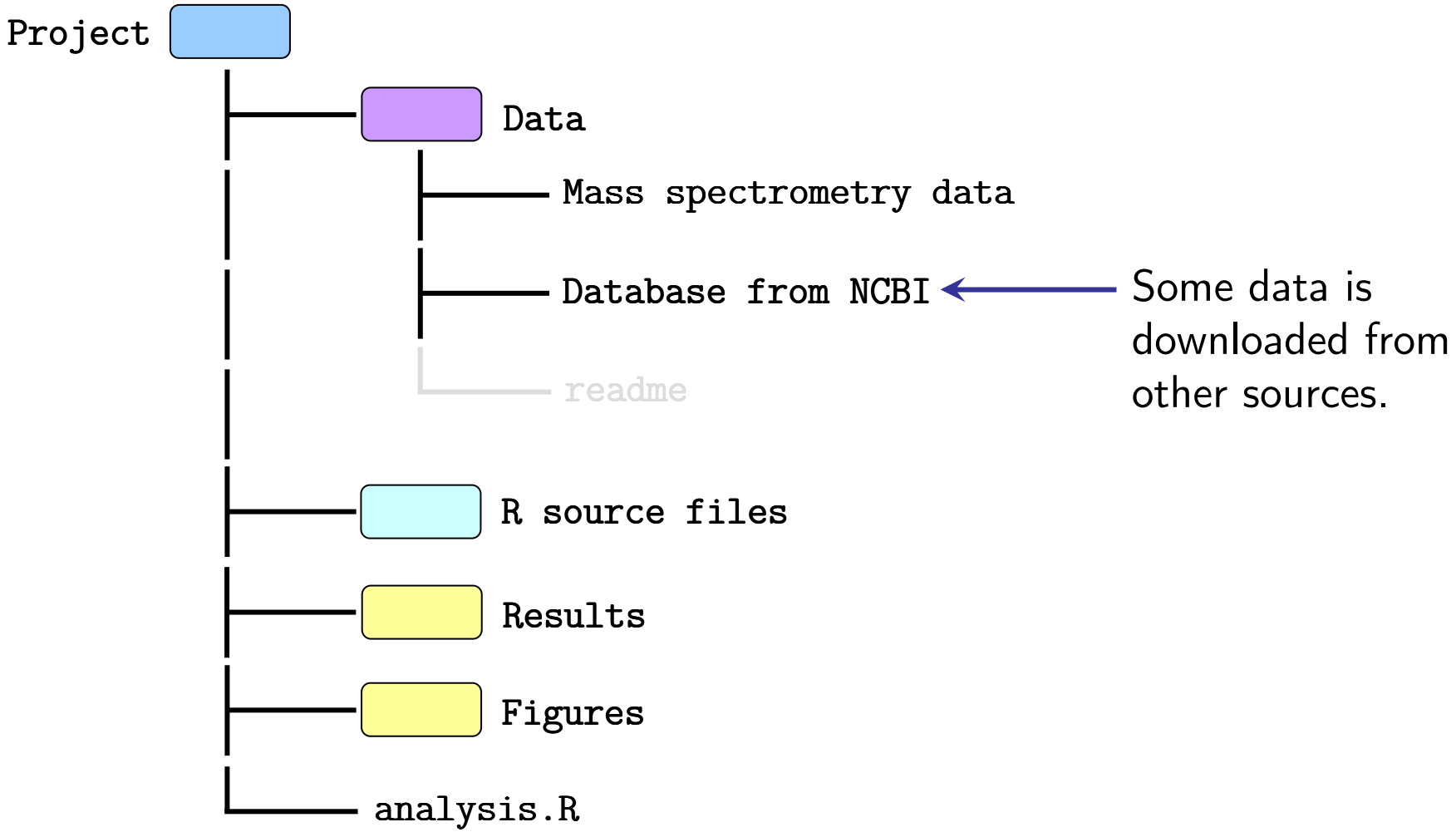
(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

Structure of a project folder



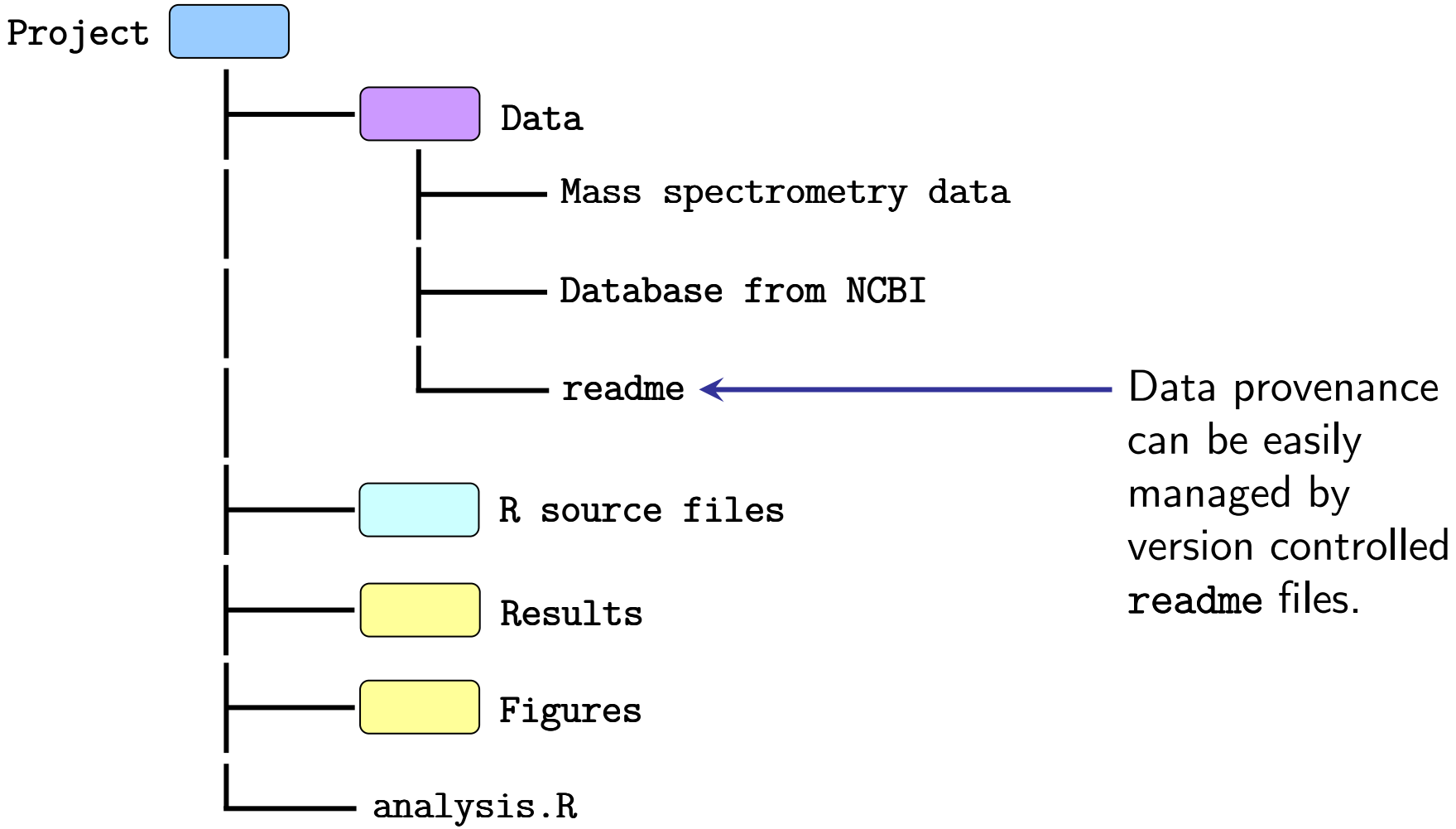
(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

Structure of a project folder



(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

Structure of a project folder



(shamelessly modified from http://nicercode.github.io/blog/2013-04-05-projects/)

Using version control with R

Version control tool: Git

- Several version control tools exist. Git is recent and reputed fast and efficient.
- R Studio can interact with Git through its graphical interface for the most basic actions.
- Git has many features accessible from the command line and can be used for any project, not just for R.

Version control tool: Git

- Several version control tools exist. Git is recent and reputed fast and efficient.
- R Studio can interact with Git through its graphical interface for the most basic actions.
- Git has many features accessible from the command line and can be used for any project, not just for R.

Version control tool: Git

- Several version control tools exist. Git is recent and reputed fast and efficient.
- R Studio can interact with Git through its graphical interface for the most basic actions.
- Git has many features accessible from the command line and can be used for any project, not just for R.

Practical:
setting up a project

Setting up a project

- R Studio can create a new empty project, or add a project environment to an existing directory.
- We are going to use an existing folder containing data and set up a project using this folder.
- The exercise folder is `Baltic_Sea_project`. Our goal is to study temporal and spatial variations of temperature in the Baltic Sea.

Setting up a project

- The `data` folder is already present in `Baltic_Sea_project`.
- We prepare the project environment by creating the other folders:
  - `R_source`
  - `figures`
  - `results`
- We will create the R files directly from R Studio.

Setting up a project

- We open R Studio and we create a project by *'Project > Create project...'* and using an existing directory (`Baltic_Sea_project`).

- We turn on the version control by *'Tools > Version control > Project Setup...'* and select Git as the version control tool.

Setting up a project

- For the first use of Git, we must specify who we are in order to correctly track who made changes.
- We must do this from the Git prompt (*'Tools > Shell...'*):

```
git config --global user.email "mdjbru@utu.fi"
git config --global user.name "Matthieu Bruneaux"
```

- Now we can create two new files (*'File > New > R Script'*) and save them as `analysis.R` and `R_source/source.R`.

Setting up a project

- Each file we want to track has to be brought to Git's attention. In the Git tab (next to History), we add `analysis.R` and `R_source/source.R` to the tracking list.

- Now is time for the first commit. We click *'Commit'* to tell Git we want to record a snapshot of the current state.

- In the *Review Changes* window, we specify a commit message before submitting the commit. Commit messages should be short and informative. For now we can just type "Initial commit".

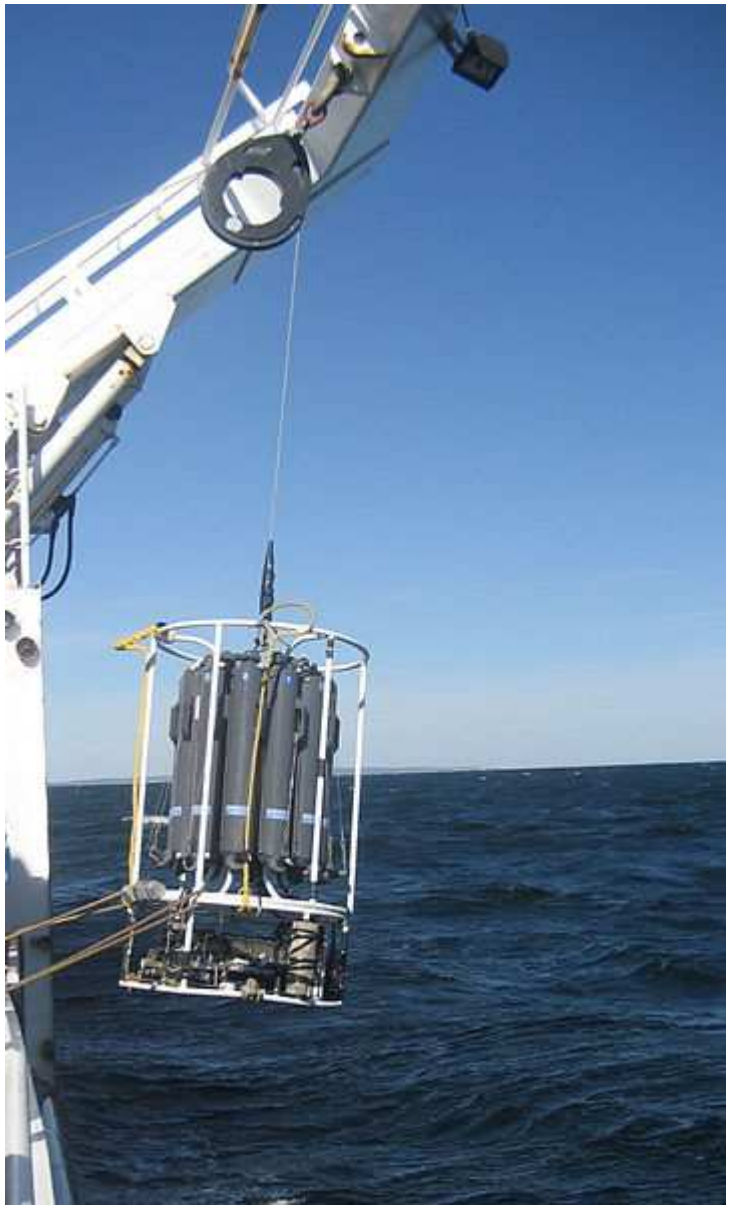
Setting up a project

- We can have a look at the Git tab again: `analysis.R` and `source.R` are no longer visible because there is no change since the last commit.
- We can also check Git history by *'More > History'* in the Git tab. There is now one commit saved in the history.
- Now let's code !

Practical:
Temporal and spatial variations of Baltic Sea temperature

Baltic Sea temperature

- Bathymetric and CTD data were downloaded from publicly available databases.
- CTD data (conductivity temperature depth) are collected in the water column by research vessels and are sometimes associated with oxygen profiles.



(from http://en.wikipedia.org/wiki/Conductivity,_temperature,_depth)

Baltic Sea temperature

- Throughout the practical, we will copy and paste chunks of code from the reference files to our working files, `analysis.R` and `source.R`.

- Let's make our first version of `analysis.R` by copying:

```
# Load the R source code
# Load the CTD data
```

- We only need to copy a function to `source.R`:

```
# Process a CTD data frame
```

R Studio shortcuts

- Run the current line: `ctrl + enter`
- Run the selection: `ctrl + enter`
- Run the previous selection again: `ctrl + shift + P`
- Run the whole script: `ctrl + shift + S`

Baltic Sea temperature

First commit

- We can save `analysis.R` and `source.R`.
- The Git tab now shows that those files are modified. When we run `analysis.R`, everything seems to be working fine.
- Let's commit the changes: in the Git tab, we stage both R files and we click *'Commit'* and as a commit message, we put "CTD data loaded and processed".

Baltic Sea temperature

History

- If we have a look at the Git history, there is now a new commit displayed.
- There is only one branch (`master`). As shown by the position of `HEAD`, we are now in the `master` branch, but there is not really any other choice for the moment.
- We can also see the differences between the two last commits for each file, but for now it actually consists of the whole of each file.

Baltic Sea temperature

New commit

- Let's add the bathymetry data.

  analysis.R
  ```
  # Load the topography data (for bathymetry)
  ```
  source.R
  ```
  # Decrease a topography grid resolution
  ```

- As always, we carefully check and test the new code .

- If we are happy, we save the files, stage them and commit the changes ("Bathymetry data loaded and grid resolution decreased").

Baltic Sea temperature

New commit

- So far so good. Let's plot the bathymetric map.

  analysis.R

```
# Draw a bathymetric map of the Baltic Sea
```

  source.R

```
# Add a color value for the sea depth to a topography
  grid
```

- The map looks good. We can commit again now ("Bathymetric map Baltic Sea").

Baltic Sea temperature

Differences

- We want to focus on the area between the Aland Islands and the Gulf of Finland.

  analysis.R

  (Note: to place before `# Draw a bathymetric map...`)

  ```
  # Filter the data for a specific area
  ```

  source.R

  ```
  # Filter a table based on coordinates
  ```

- Let's commit ("Area restricted to central Baltic Sea"). If we look at the differences for each file now, we can see that Git shows precisely where the insertions occurred.

Baltic Sea temperature

Differences

- We changed our mind and we want to look at the northern part of the Gulf of Bothnia. Let's use those coordinates for the area limits:

```
x_lon_limits = c(19.3, 26.9) # North of Gulf of Bothnia
y_lat_limits = c(62.7, 66.1)
```

- Let's commit ("Area restricted to North of Gulf of Bothnia"). Only `analysis.R` was changed this time. Again, Git is good at underlining the differences clearly within a file. This is a very handy feature that gives you fine control over your changes.

Baltic Sea temperature

Reverting to a previous version

- Wait, finally the Central Baltic is probably more interesting... Let's revert back to the previous version.
- To revert `analysis.R` to its previous state, we have to go to the shell and type one of those:

```
git checkout HEAD~1 analysis.R
git checkout <commit-hash> analysis.R
```

- `HEAD~n` means the n^{th} state before the current. The hash of each commit is contained in the SHA field in Git history.

Baltic Sea temperature

Branching

- We want to explore if something interesting can come from surface temperature evolution over the last decades.
- We create a new branch for this (from the shell):

```
git branch surface_temperature
```

- We now switch to this branch:

```
git checkout surface_temperature
```

- In real life we wouldn't need a whole branch for such a simple exploratory analysis.

Baltic Sea temperature

New commit

- Let's add more code:

  source.R

```
# Filter a table based on depth and date
# Calculate average temperature per year for depth and
  date ranges
```

  analysis.R

```
# Evolution of summer surface temperature
```

- We commit the changes ("Surface temperature temporal trend").

- Actually, it turns out there is not enough data for a long-term analysis.

Baltic Sea temperature

Changing branch

- We now want to go back to the master branch, and from there start a new branch looking at variation of temperature with depth.

- From the shell:

```
git checkout master
git branch temperature_depth
git checkout temperature_depth
```

Baltic Sea temperature

New commit

- Let's add more code:

  analysis.R

```
# Temperature profile in relation with depth
```

- We commit the changes ("Temperature profile in relation with depth").

- If we look at Git history and select *'(all branches)'*, we can see how the project is evolving.

Baltic Sea temperature

New commit

- The previous plot showed nicely how temperature is more stable below 50m. However, we are using temperature from all seasons. Let's correct this and compare winter and summer profiles.

- Let's add more code:

  analysis.R

  replace

  ```
  # Temperature profile in relation with depth
  ```

  with

  ```
  # Seasonal temperature profile in relation with depth
  ```

  source.R

  ```
  # Filter a table based on year day
  ```

Baltic Sea temperature

New commit

- We commit ("Temperature profile plotted for winter and summer")
- The seasonal plots show clearly the different patterns between the surface and the deeper water.
- Now let's try to improve a bit our bathymetric map.

Changing branch

- We now want to go back to the master branch, and from there start a new branch focusing on the bathymetric map.

- From the shell:

```
git checkout master
git branch bathymetric_map_ggplot2
git checkout bathymetric_map_ggplot2
```

Baltic Sea temperature

New commit

- Let's add more code:

  analysis.R

```
# Bathymetric map with ggplot2
```

- We commit the changes ("Bathymetric map with ggplot2").
- We can now see the three branches in Git history.
- We were really over-enthusiastic about branching here.

Baltic Sea temperature

New commit

- Let's add the location of the CTD on the map:

  analysis.R

```
# Draw the CTD locations on the map
```

- We commit the changes ("CTD locations added to the map").

Baltic Sea temperature

Merging branches

- Now we are happy with the map and we would like to merge the branches `temperature_depth` and `bathymetric_map_ggplot2` to have everything in one script.

- From the shell:

```
git merge temperature_depth
```

- Conflicts might arise when there is ambiguous file changes. We have to solve them manually.

Solving conflicts

- Git tries to merge as much as it can and shows remaining conflicts with <<<<< and >>>>> in the file.
- We try to solve them manually, then commit ("Successful merge").
- If it is too hard to do, we cancel the merge (shell) and rethink our strategy:

```
git merge --abort
```

- In real life, single-user projects rarely need branching.

Baltic Sea temperature

Generating figures

- We can save the temperature profiles in our `figures` folder:

  `analysis.R`

  ```
  # Save figures for temperature profiles
  ```

- Commit ("Figures for temperature profiles").

- It would be nice to keep track of the Git commit hash to know which version generated which figure.

Baltic Sea temperature

Generating figures

- A bit of code can do this:

  analysis.R

  replace

  ```
  # Save figures for temperature profiles
  ```

  with

  ```
  # Save figures for temperature profiles with Git hash
  ```

  source.R

  ```
  # Get current Git hash
  ```

- Commit ("Figures tagged with Git hash").

Baltic Sea temperature

Generating figures

- Let's add some color with the parameters `col="blue"` and `col="red"` added to the plot commands for the winter and summer profiles, respectively.
- Commit ("Color added to the temperature profiles").
- If we generate the figures again, the Git hash is also updated. This means we can now reproduce any figure we made at any point, as long as a commit was done before generating the figure.

Baltic Sea temperature

Generating tables

- We can save the temperature profile values in our `results` folder:

  `analysis.R`

  ```
  # Save tables for temperature profiles
  ```

- Commit ("Tables for temperature profiles").

- Now our table output is also traceable using the Git hash.

Baltic Sea temperature

Tagging a commit point

- When we reach a milestone in our analysis development, we want to be able to mark it and retrieve it easily in the commit flow.

- We can tag it with Git (from the shell):

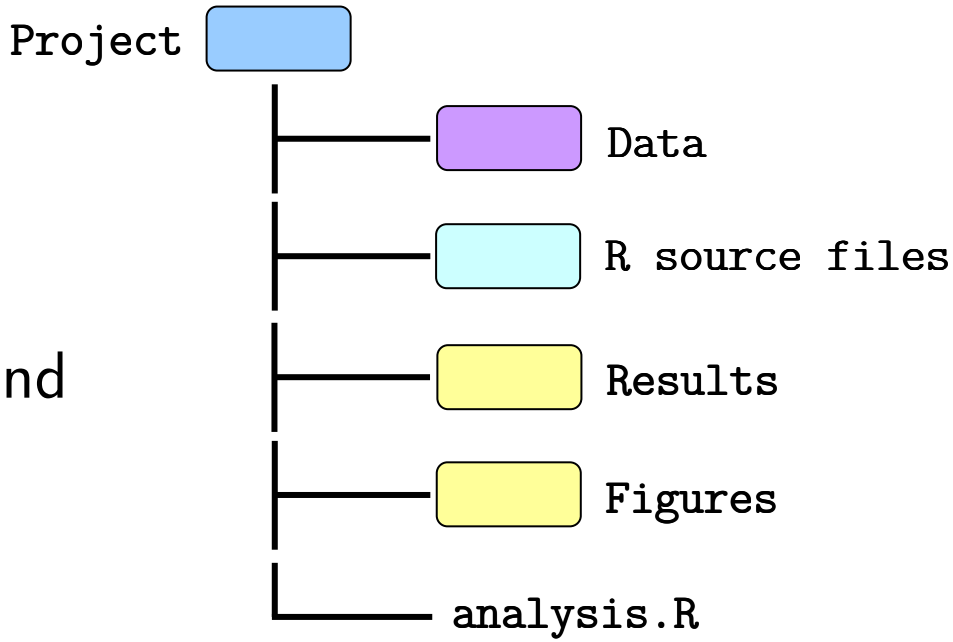
```
git tag -a v1.0 -m "Figures and tables of the temperature
profiles in relation with depth for the central Baltic Sea,
separated by season"
```

- The format for tagging, getting the list of tags and showing a tag information is:

```
git tag -a <tag_name> -m <tag_comment>
git tag
git show <tag_name>
```

Structure of a project folder

Clear and run again



- Our project is now version controlled and every figure and table produced is traceable.

- We can generate a clean set of figures and tables by deleting the contents of `figures` and `results`, and running again `analysis.R`.

Further reading and references

Further reading and references

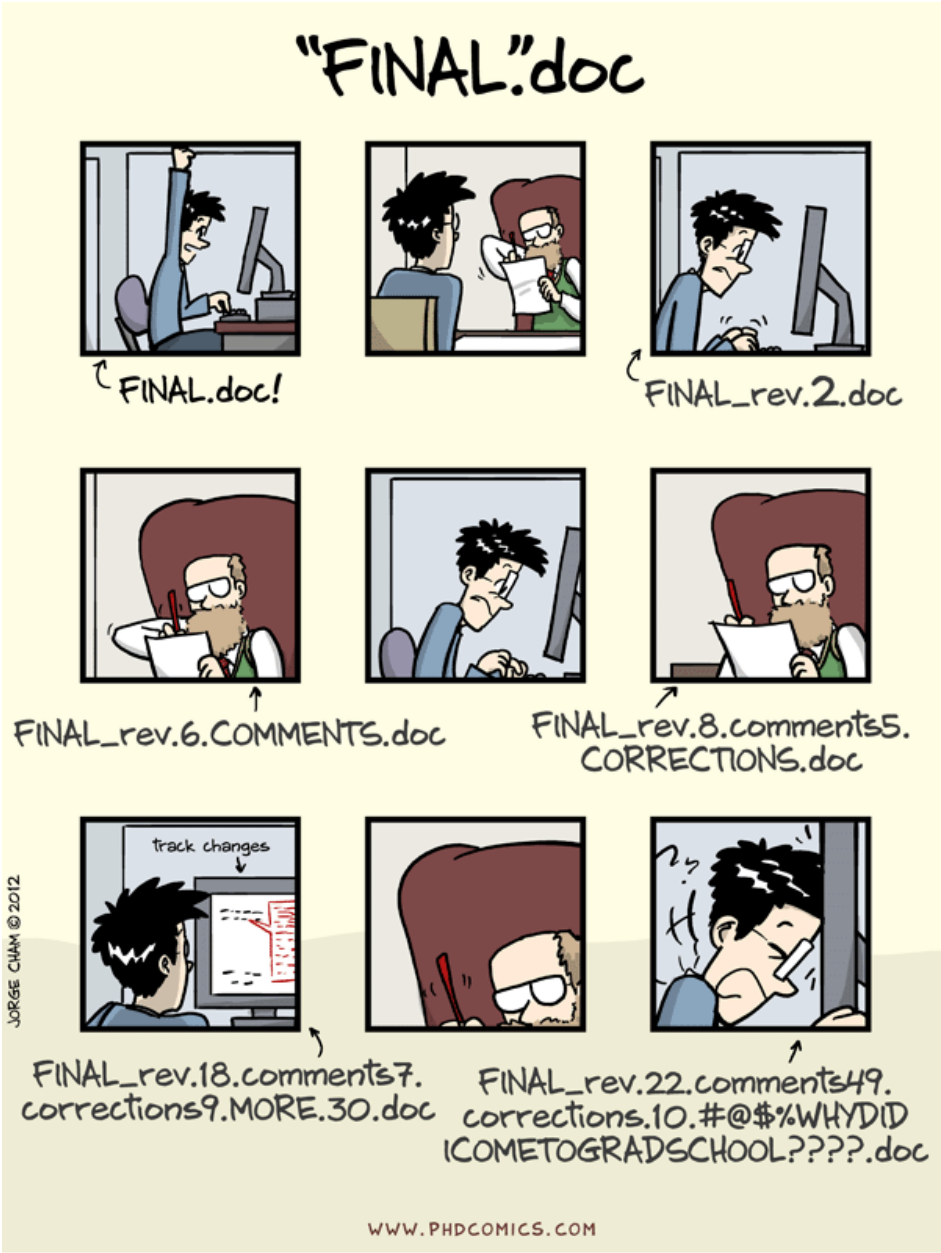
Reproducibility of analysis, data management

- Project design:
  http://nicercode.github.io/blog/2013-04-05-projects/
- Data management:
  http://software-carpentry.org/4_0/data/mgmt.html

Using Git

- Using Git with R Studio:
  http://nicercode.github.io/git/
- Git reference:
  see the previous link and http://git-scm.com/documentation

A common problem



Next Part
Report generation