# EL4: Tooling

## Positron and version control

# History

> It is widely acknowledged that the most fundamental developments in statistics in the past 60 years are driven by information technology (IT). We should not underestimate the importance of pen and paper as a form of IT but it is since people start using computers to do statistical analysis that we really changed the role statistics plays in our research as well as normal life.

> Although: “Let’s not kid ourselves: the most widely used piece of software for statistics is Excel.”” /Brian Ripley (2002)

Short overview

## General-Purpose Programming Languages

Early statistical computing relied heavily on:

- **FORTRAN**
  - Dominant language for numerical and statistical computation
  - Statistical methods implemented as libraries and subroutines
  - Still used as subroutines in modern statistical software!
- **ALGOL / ALGOL 60**
  - Used mainly in academic environments
- **PL/I**
  - Used in some government and industrial contexts

📌 These languages required substantial programming expertise.

## Early Statistical Packages

Several dedicated statistical systems emerged:

- **SPSS** (1968)
  - Originally batch-oriented
  - Widely used in social sciences
  - Originaly “Statistical Package for the Social Sciences”
- **BMDP** (1960s)
  - Developed at UCLA
  - Common in medical statistics

  - Bio-Medical Data Package
- **GENSTAT** (1968)
  - Focused on agricultural statistics
- **MINITAB** (1972)
  - Designed for teaching and education
  - Still popular in quality control

## SAS: A Transitional System

- **SAS** (early 1970s)
- Developed for agricultural and biostatistical analysis
- Script-based, but largely batch-oriented
- Became a standard in:
  - Government agencies
  - Large organisations
- Known for strong data management capabilities
- Still widely used in pharmaceutical industry and clinical trials

📌 SAS predates S but influenced later statistical workflows.

## Limitations of Pre-S Systems

Common limitations included:

- Batch processing rather than interactivity
- Separation of data management and analysis
- Limited graphics capabilities
- High barriers to exploratory data analysis

## S

- S takes form at Bell Laboratories (interactive statistical computing).
- John Chambers leads the effort.
- **1976:** first working version of S runs on GCOS
- **1979:** S2 is ported to UNIX; UNIX becomes the primary platform
- **1980:** S is first distributed outside Bell Labs
- **1981:** source versions are made available
- **1984:** key S books published (often called the “Brown Book” era)

## The New S Language

- **1988:** “New S” is released (major language redesign)
- **1988:** **S-PLUS** is first released as a commercial implementation of S
- **1991:** *Statistical Models in S* (“White Book”) popularizes formula notation (the ~ operator), data frames, and modeling workflows

## R

- **1993:** first versions of **R** are published (Auckland; Ross Ihaka & Robert Gentleman)
- **1995:** R becomes open source (GPL)

- **1997:** the R Core group forms; **CRAN** is founded (Kurt Hornik)
- **2000: R 1.0.0** is released 2000-02-29

### RStudio brings an IDE to the R community

- **2009:** RStudio (the company) is founded
- **2011:** RStudio IDE is introduced as an open-source IDE for R (desktop + server)

### Microsoft (Revolution Analytics)

- **Jan 2015:** Microsoft announces it will acquire **Revolution Analytics**
- Microsoft promotes enterprise R offerings (e.g., Microsoft R Open / R Server)
- **2016:** SQL Server 2016 introduces **R Services** (in-database R)
- **2017:** Microsoft expands the stack under "Machine Learning Server" branding
- **June 2021:** Microsoft announces retirement of **Microsoft Machine Learning Server**
- **July 1, 2023:** Microsoft era over

### RStudio becomes Posit

- **July 27, 2022:** RStudio rebrands as **Posit**
- **July 28, 2022: Quarto** is announced as a next-generation scientific and technical publishing system (multi-language, multi-engine)

## Modern IDE

### Positron

- New generation IDE for data science
- From Posit PBC
- Free for individual use
- Based on Code OSS (open source version of VS Code from Microsoft)
- For both R and Python (Julia?)

### Quick tour

> ! Positron assistant (mentioned in the video)
>
> This feature will most likely be disabled in any secure working environment. Such environments often have strict rules about data privacy and security, which may conflict with the assistant's functionality. Health data in SENSITIVE and SECURE environments must not be shared with external services, including AI assistants, to comply with data protection regulations and institutional policies.
>
> It is recommended to not rely on such tools during the course (even if all our data is synthetic). If you start to rely on such tools, you might get difficulties the day you work with real data (might lead to prosecution for "brott mot tystnadsplikten" which is not only public, but actually civil law ("Brottsbalken") with prision sentence as a possibility). Society put an extreme emphasis on protecting health data, and rightfully so!

# Version control

## Before Version Control Systems

1950s–1970s: Early software development relied on:

- Manual file naming:
  - `analysis_final.f`
  - `analysis_final_v2.f`
- Physical media:
  - Punch cards
  - Magnetic tape
- Centralised mainframes

📌 No automated tracking of changes.

## Floppy discs, Mail, and Shared Directories

1970s–1980s: Common practices included:

- Copying files to:
  - Floppy disks
  - Magnetic tapes
- Sending media by **postal mail**
- Sharing files via:
  - Network drives
  - FTP servers

📌 Version control was social, not technical.

## Early Version Control Systems

1980s: First-generation tools focused on single files:

- **SCCS** (Source Code Control System, 1972)
- **RCS** (Revision Control System, 1982)

Characteristics:

- Versioning per file
- Linear history
- Central storage

## Centralised Version Control

1990s: Project-level systems emerge:

- **CVS** (Concurrent Versions System)
- **Subversion (SVN)**

Key features:

- Central repository

- Multiple users
- Check-in / check-out model

📌 Still required constant access to the central server.

## Limitations of Centralised Systems

Common problems:

- Single point of failure
- Poor support for branching and merging
- Difficult offline work
- Slow operations on large repositories

These limitations became critical for large projects.

## Git

- **Git** is created by Linus Torvalds in 2005
- Original motivation:
  - Support Linux kernel development
  - Replace proprietary tools

Design principles:

- Distributed architecture
- Fast local operations
- Strong support for branching and merging

## Distributed Version Control with Git

Key ideas in Git:

- Every clone is a full repository
- Local commits without network access
- Cheap and fast branches
- Cryptographic integrity (hash-based)

📌 Collaboration becomes more flexible and robust.

## Hosting Platforms

Platforms built around Git:

- **GitHub** (2008)
- **Bitbucket** (2008)
- **GitLab** (2011)
- Gitea - open source alternative to GitHub (get the same functionallity locally or on a server)

They add:

- Pull requests / merge requests
- Issue tracking

- Code review
- CI/CD integration

## Version Control Today

Modern usage includes:

- Code
- Documentation
- Data analysis (scripts, notebooks)
- Configuration and infrastructure

Git is integrated into:

- IDEs (VS Code, Positron)
- CI/CD pipelines
- Cloud platforms

## Version Control Beyond Code

Today, version control supports:

- Reproducible research
- Collaborative writing
- Data science workflows
- Teaching and learning

📌 Version control is now a core professional skill.

## WARNING!

- Not everything should be shared!
- Scripts and documentation yes!
- But **Health data is sensitive!**
  - Do not share it!
  - Avoid unintentional sharing!
  - Private repositories are still shared with hosting provider!
- Avoid explicit file paths and sensitive info in scripts!
- Can give information about data location and internal structure!
- No hard-coded passwords or API keys!

## Git basics

(After installing the Git software)

- Collect all files related to a project in a folder
- Initialize a git repository in that folder
- Make changes to files
- Stage changes for commit
- Commit changes with a message
- Possibly push commits to remote repository

```
cd path/to/your/project
git init
git status
# make changes to files
git add filename1 filename2
git commit -m "Descriptive message about changes"
git remote add origin
```

### Video tutorials

The video below is a good start to understand the basic concepts of Git and GitHub (and there are others to be found on YouTube). Note that there are a lot of videos on Git combined with Rstudio. They might still be relevant even though we are using Positron here.

### Inspirational video

Watch this video even though some parts might be overwhelming. It gives a good overview of the current state (2025), even though many things will be too advanced for this course (it is not specifically aimed for statisticians or R users).

> ! .gitignore
>
> The `.gitignore` file is very important in settings with health data! Pay close attention to this section of the video!

### Git in Positron

- Remember that Positron is build on Code OSS (which shares a lot of features with VS Code).
- Branching and merging is possible but we will not cover that here.

Short official introduction from Microsoft:

More detailed introductions. Watch both! The first one is based on a Windows version of VS code and the second on Mac but the concepts are the same:

> i Overwhelmed?
>
> This video includes some parts which might be overwhelming if you are new to Git and GitHub. Don't worry! You don't need to understand everything right away. Just try to follow along with the basic concepts and steps. You will get more comfortable with practice.

## Statistics projects

### Not a single R script

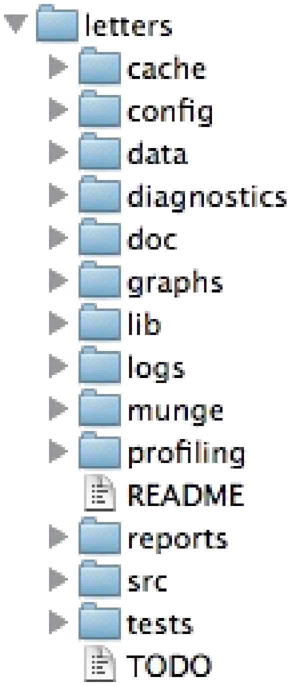
- Real projects are more complex than a single R script!
- Multiple scripts

- Data files
- Documentation
- Reports
- Version control
- Reproducible workflows

## Project structure

Common file structures

- Help you organize your thoughts
- Help others to collaborate
- simplifies paths used in your code



letters
cache
config
data
diagnostics
doc
graphs
lib
logs
munge
profiling
README
reports
src
tests
TODO

## Example structure

```
/.../my_project/
  ├── README.md         - project documentation
  ├── TODO              - what should be done next?
  ├── .git              - handled by git (hidden folder)
  ├── .gitignore        - used by git but your responsibility!
  ├── data/             - your data files (not under version control!)
      ├── cancer.csv
      └── patients.qs
  ├── R/                - your saved R functions
      ├── function1.R
      └── function2.R
```

```
   ├── reports/
   └── _targets.R        - targets pipeline script
```

#### `README.md`

- Document the purpose of the project
- What is it about?
- What is the aim?
- Who to contact for questions?
- In what circumstances was it created?

Markdown format (simple text with some possible formatting)

### `data` folder

- Store your data files here as they are when you get them
- Avoid any modifications to the raw files!
- It is very easy to forget what you do if it can not be traced by code
- Do NOT include this folder in version control!
- Git is not good at handling large files
- Sensitive data should not be shared!
- Add `data/*` to your `.gitignore` file
- In realistic projects, data might come in varying formats
  - csv, txt, xlsx, sas7bdat, sav, dta, etc
  - some files might be very big (gigabytes not uncommon)

### `R` folder

- Store your R functions here
- Document their purpose inline!
- Helps you to reuse code
- Easier to read main scripts
- Easier to test and debug code
- Easier to share code between projects

### `reports` folder

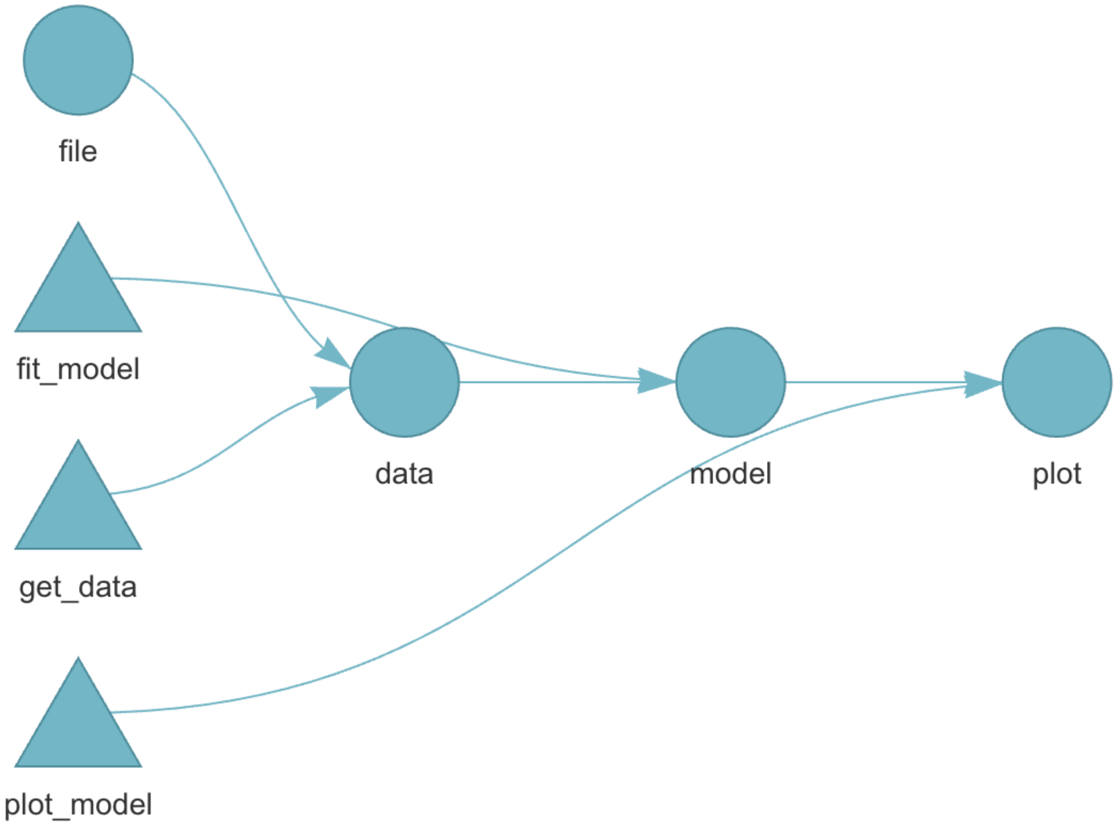
- Store your reports here
- Quarto documents
- Documemnt your analysis
- Include figures and tables
- Share with external collaborators

### `_targets.R` file

Use the `targets` package to create a reproducible data analysis pipeline

## Pipeline

- Define steps in your analysis as targets
- Define dependencies between targets
- Automatically track changes and rerun only necessary parts



## Why?

- You do not want to rerun everything all the time!
- You want to keep track of what you have done
- You want to be able to reproduce your results later
- You want to share your workflow with others

## Reading

- A bit old but still relevant: https://happygitwithr.com/
- Targets overview
- Target manual