Methods for unstructured data

Lecture 1: Tools for Scientific Programming

Helge Liebert

What is this lecture about?

- Most research in economics now involves scientific programming.
- Introduce tools and ideas that may make daily research tasks easier.
- Many of these have been developed by other scientists or IT professionals. Some are decades old, others are novel.
- Focus on data processing and reproducible research.
- Text processing is a common task in research and industry.
- Working with text data requires understanding of some practical techniques.

Contents

Economics and computation

Economics and computer science

- Technical aspects rarely part of the economics curriculum.
- Data management is not taught in introductory econometrics.
- Computer science often involves processing data.
- Problems you are likely to encounter have been solved.
- Scientific programming and “big data” were common in other sciences before they gained a more prominent role in economics.
- Tools and concepts that economists can profit from.
  - Remote servers, databases, version control, acessing APIs for data, text processing and analysis, geospatial analysis, OCR, automation, ...
  - Time complexity of algorithms, computational cost, databases, ...

Survey

- Operating systems - Windows, MacOS, Linux?
- Scientific programming - Stata, R, Python, Matlab/Octave, Julia, ...?
- General purpose programming languages?
- Servers, shell scripting, CLI?
- (Relational) databases - SQL?
- Version control - Git?

Topics

- General points about scientific programming and working with computers.
- Superficial and incomplete. A stand-alone course on Programming Practices for Research could be devoted to these topics.
- Brief intro to get you going and help identify appropriate tools for a problem.
- Point you towards resources and explain their general concepts.
- Leave you marginally more computer literate.

- Languages, tools, version control.
- Unix shell and the command line.
- Regular expressions.

Programming languages

Which language to choose?

- *It depends.* Choice is use-case- and taste-specific.
- *Anything* can be done in *any* language. Convenience varies.
- Concepts and toolkits transfer easily most of the time.
- Trade-off: Prior knowledge vs. task suitability.
- Languages which are great for specific scientific purposes may not be well-suited for other tasks.
- Never re-invent the wheel.

Possible options

- Specialized languages: R, MATLAB/Octave, Stata, Gauss, Julia, ...
- General-purpose languages: Python, Perl, Ruby, C, ...
- Choose a high-level, dynamic, interpreted language unless you are sure you require the extra speed of a compiled language.
- Ideally free and open source. Popular is typically better.

- Research ex ante which libraries are most mature and best for solving your specific problem.
- Focus on getting things done. Utilize prior knowledge.

This lecture

- Any task covered by this lecture can be accomplished using R or Python (augmented by shell programs). The lab sessions mainly rely on R.
- R, Python, SQL and knowing your way around a terminal are highly valued skills on the job market.
- Rule-of-thumb recommendation:
  - Simple data analysis/small text corpara:
    Stick with R. Augment with other tools where required.
  - More involved data processing/larger text corpora:
    Go with Python. You can still analyze data in R.
  - ... and whatever program your colleagues are using.

R

- R is the major statistical programming language.
- It is free, used in many sciences and in industry. Good documentation.
- New models are frequently published and implemented first in R.
- Having data processing and analysis in the same language is nice.
- Good library support for common tools (e.g. databases, regular expressions).
- Specific tasks for which high-level wrapper functions are not available may be very cumbersome.
- In recent years, R development has been very active and libraries exist for almost anything.

Python

More

- General-purpose programming language, supports object-oriented programming.
- Reads like english. Explicit and clear. Whitespace matters, no braces. (*“There should be one obvious way to do it”*.)
- Used extensively in industry and sciences. Good documentation.
- Libraries for almost anything.
- Many science-related libraries exist for other languages, but rarely are they as mature.
- Growing support for statistical modeling.
- A bit less suited for interactive data work (but more so for deployment in production).

Why not Stata, Matlab, Gauss or similar?

- Advantage: Many domain-specific models supported.
- Less support for almost anything else.
- Much less flexible for anything not to do with data analysis or numerics.
- Difficult to deploy on a server. Often tied to a GUI.
- Less popular, smaller userbase. Proprietary and expensive.
- You can still rely on them for estimation after your data is clean.

Why not Perl or Ruby?

Perl

- *“There’s more than one way to do it.”*
- Lots of special cases, reliance on hidden magic, bad readability.
- You may want to work together with somebody else.
- You may want to understand your own code in a few months time.
- Less popular in sciences.

Ruby

- Everything is an object. Intuitive.
- Different focus.
- Even less popular in sciences.
- Less support.

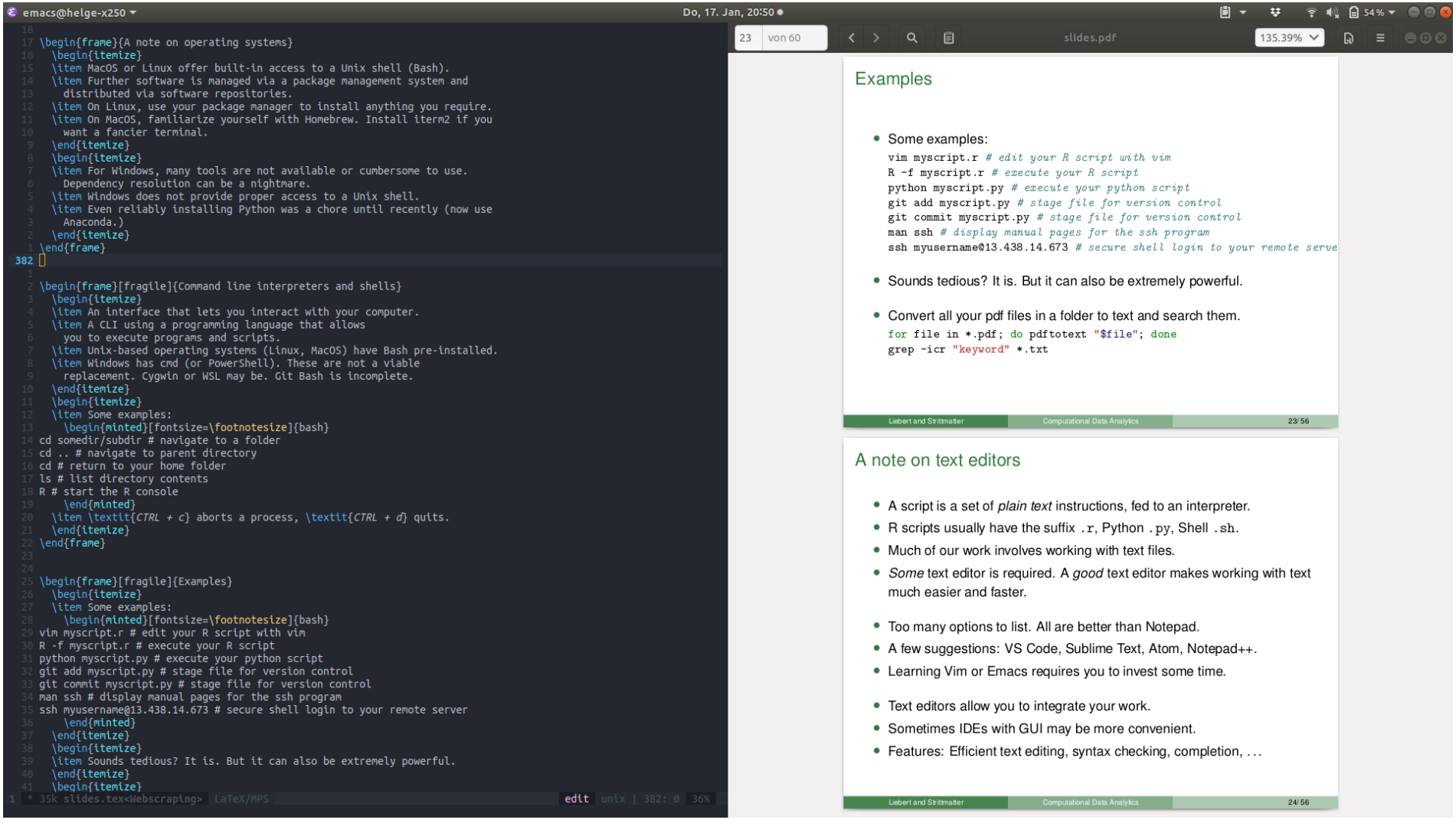
A note on text editing

- A script is a set of *plain text* instructions, fed to an interpreter.
- Editing is independent from running code.
- R scripts usually have the suffix `.r`, Python `.py`, Shell `.sh`.
- Proficiency in a text editor makes working with text easier and faster.

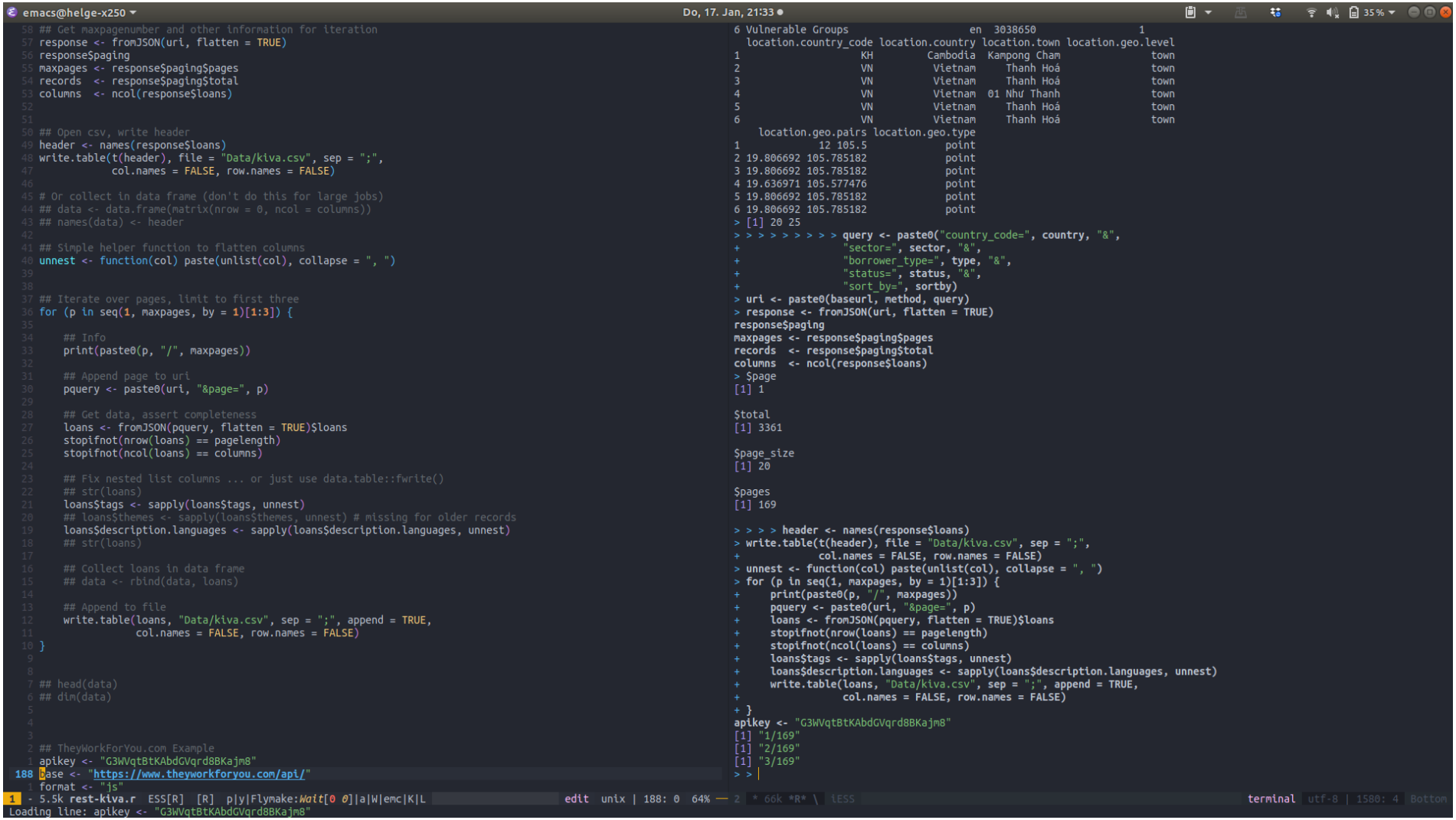
- Too many options to list. All are better than Notepad.
- A few options: VS Code, Sublime Text, Atom, Notepad++, ...
- Learning Vim or Emacs requires you to invest some time.

- Text editors allow you to integrate your work and edit text efficiently.
- Sometimes IDEs with GUI may be more convenient.
- Features: Regex search and replace, diff, syntax checking, formatting, completion, documentation lookup ...

A possible setup



… that is universal



… that is universal

```
each school-track, classroom formation is conditionally ignorable with respect to SN
status, as students from different primary school districts are mixed and their SN status
is not observed by secondary school administrators.\footnote{Using PISA data from
secondary schools in Switzerland, \citet{Vardardottir2015} shows that track-by-school
fixed effects render peer group composition conditionally uncorrelated with a large set
of students' characteristics, while track fixed effects and school fixed effects do not.}
Neither primary schools nor the SPS share information with secondary schools for equity
reasons and to avoid stigma when transitioning between schools.
174
175 % NEU: BEu bitte lesen
176 % changed some small things. ok for me.
177 Beyond this anecdotal evidence, we formally test the validity of the identification
strategy with four balancing tests, which are presented and discussed extensively in
Appendix B. First, we examine whether the proportion of SN peers predicts individual
baseline characteristics (gender, native speaker, and age). The aim of this test is to
detect potential selection into classrooms. We also conduct this test separately for SN
and non-SN students. None of the baseline characteristics are statistically significant
at conventional levels, either considered individually or jointly. Second, we regress the
indicator for SN status on class fixed effects, which should be jointly insignificant if
assignment to classrooms is ignorable with respect to SN status \citep{chettyEtal2011}.
We also conduct this test to check for ignorable assignment of SN students to teachers.
We find no evidence for systematic assignment of SN students to either classes or
teachers. Third, we conduct a simulation exercise in the spirit of \citet{carrell2010}.
We re-sample classes and thereby assign SN students randomly to classrooms, and test
whether the observed distribution of SN students differs from the simulated one. In
addition, we compute the interquartile range of the proportion SN students across classes
for each simulation, and compare these simulated interquartile ranges with the one we
observe in the data. Neither simulation procedure uncovers any worrisome pattern in the
assignment of SN students to classes. Fourth, we decompose the variation in the fraction
of SN peers across and within schools. To do so, we examine the residual variation in the
proportion of SN peers after partialling out the school-track-year fixed effects. We find
that the residual distribution in the proportion of  SN peers is consistent with
variation from a random process. Overall, the balancing tests we performed indicate that
the key identification assumption of (conditionally) ignorable assignment of SN students
to classes is plausible.
178
179 Our identification relies on variation between classes within school-track-years.
Although families can potentially choose their district of residence and thereby
influence schooling options for their children, possible selection into schools does not
confound our results.\footnote{Endogenous class formation could still occur if parents
request to transfer their children to a class with a lower SN fraction. To investigate
this potential threat, we acquired the official education statistics from the Swiss
Federal Statistical Office (SDL, \textit{Statistik der Lernenden} in German) for the
years 2012-2015. Importantly, the SDL has a classroom ID which allows us to reconstruct
the classes within each school-year-track. For the state of St.\ Gallen, we find that on
A:  ●  104k  xSource/manuscript_R1_v3.tex  ??:0 37%        LF UTF-8  LaTeX/FPS  ⊘ 1

different education tracks, and classes are strictly separated between tracks. Within
each school-track, classroom formation is conditionally ignorable with respect to SN
status, as students from different primary school districts are mixed and their SN status
is not observed by secondary school administrators.\footnote{Using PISA data from
secondary schools in Switzerland, \citet{Vardardottir2015} shows that track-by-school
fixed effects render peer group composition conditionally uncorrelated with a large set
of students' characteristics, while track fixed effects and school fixed effects do not.}
Neither primary schools nor the SPS share information with secondary schools for equity
reasons and to avoid stigma when transitioning between schools.
172
173 % NEU: BEu bitte lesen
174 Beyond this anecdotal evidence, we formally test the validity of the identification
strategy with four balancing tests, which are presented and discussed extensively in
Appendix B. First, we examine whether the proportion of SN peers predicts individual
baseline characteristics (gender, native speaker, and age). The aim of this test is to
detect potential selection into classrooms. We also conduct this test separately for SN
and non-SN students. None of the baseline characteristics are statistically significant
at conventional levels, either considered individually or jointly. Second, we regress the
indicator for SN status on class fixed effects, which should be jointly insignificant if
assignment to classroom is ignorable with respect to SN status \citep{chettyEtal2011}. We
also conduct this test to check for ignorable assignment of SN students to specific
teachers. We find no evidence for systematic assignment of SN students to either classes
or teachers. Third, we conduct a simulation exercise in the spirit of
\citet{carrell2010}. We re-sample classes, randomly assign SN students to classes, and
test whether the observed distribution of SN students differs from the simulated one. In
addition, we simulate random classroom assignment within schools, compute the
interquartile range of the proportion SN across classes, and compare the simulated
interquartile range with the one we observe in the data. Neither simulation procedure
uncovers any worrisome pattern in the assignment of SN students to classes. Fourth, we
decompose the variation in the fraction of SN peers across and within schools. To do so,
we examine the residual variation in the proportion of SN peer after partialling out the
school-track-year fixed effects. We find that the residual distribution in the proportion
of  SN peers is consistent with variation from a random process. Overall, the balancing
tests we performed indicate that the key identification assumption of (conditionally)
ignorable assignment of SN students to classes is plausible.
175
176 Our identification relies on variation between classes within school-track-years.
Although families can potentially choose their district of residence and thereby
influence schooling options for their children, possible selection into schools does not
confound our results.\footnote{Endogenous class formation could still occur if parents
request to transfer their children to a class with a lower SN fraction. To investigate
this potential threat, we acquired the official education statistics from the Swiss
Federal Statistical Office (SDL, \textit{Statistik der Lernenden} in German) for the
years 2012-2015. Importantly, the SDL has a classroom ID which allows us to reconstruct
the classes within each school-year-track. For the state of St.\ Gallen, we find that on
B:  ●  104k  xOld/manuscript_R1_v2.tex  ??:0 37%        LF UTF-8  LaTeX/FPS  ⊘ 11

Type ? for help
-- *Ediff Control Panel*   diff 4 of 15        Quick Help
Auto-refining is ON
```

Rstudio

RStudio

Haiyan-13.R

```
1  library(XML)
2  url <- "http://www.gdacs.org/Cyclones/report.aspx?eventid=41058&episodeid=28&
3  dat <- readHTMLTable(readLines(url), which=5)
4  dat$latlon <- dat[,8]
5  levels(dat$latlon) <- sapply(
6    strsplit(levels(dat[,8]), ",\n        "),
7         function(x) paste(x[2], x[1], sep=":")
8  )
9  dat$Category <- factor(dat$Category, levels=levels(dat$Category)[c(6,7,1:5)],
10                        ordered=TRUE)
11 dat$cat <- as.numeric(dat$Category)
12 dat$Gust_kmh <- dat[,6]
13 levels(dat$Gust_kmh) <- sapply(strsplit(levels(dat[,6]), "km"),
14                                function(x) gsub(" ", "",x[1]))
15 dat$Gust_kmh <- as.numeric(as.character(dat$Gust_kmh))
16 M <- gvisGeoChart(dat, "latlon", sizevar="cat",
17                   colorvar="Gust_kmh",
18                   options=list(region='035',
19                                backgroundColor="lightblue",
20                                datalessRegionColor="grey"))
21 plot(M)
22
```

10:37 (Top Level) R Script

Console ~/Downloads/IdR-dev/IdR/

```
> dat$latlon <- dat[,8]
> levels(dat$latlon) <- sapply(
+   strsplit(levels(dat[,8]), ",\n        "),
+         function(x) paste(x[2], x[1], sep=":")
+ )
> dat$Category <- factor(dat$Category, levels=levels(dat$Category)[c(6,7
,1:5)],
+                        ordered=TRUE)
> dat$cat <- as.numeric(dat$Category)
> dat$Gust_kmh <- dat[,6]
> levels(dat$Gust_kmh) <- sapply(strsplit(levels(dat[,6]), "km"),
+                                function(x) gsub(" ", "",x[1]))
> dat$Gust_kmh <- as.numeric(as.character(dat$Gust_kmh))
> M <- gvisGeoChart(dat, "latlon", sizevar="cat",
+                   colorvar="Gust_kmh",
+                   options=list(region='035',
+                                backgroundColor="lightblue",
+                                datalessRegionColor="grey"))
> plot(M)
>
```

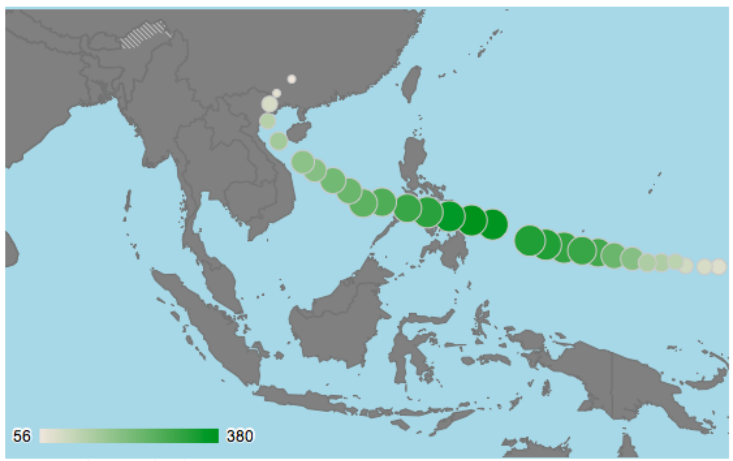
Environment | History | Presentation

Import Dataset | Clear | List

Global Environment

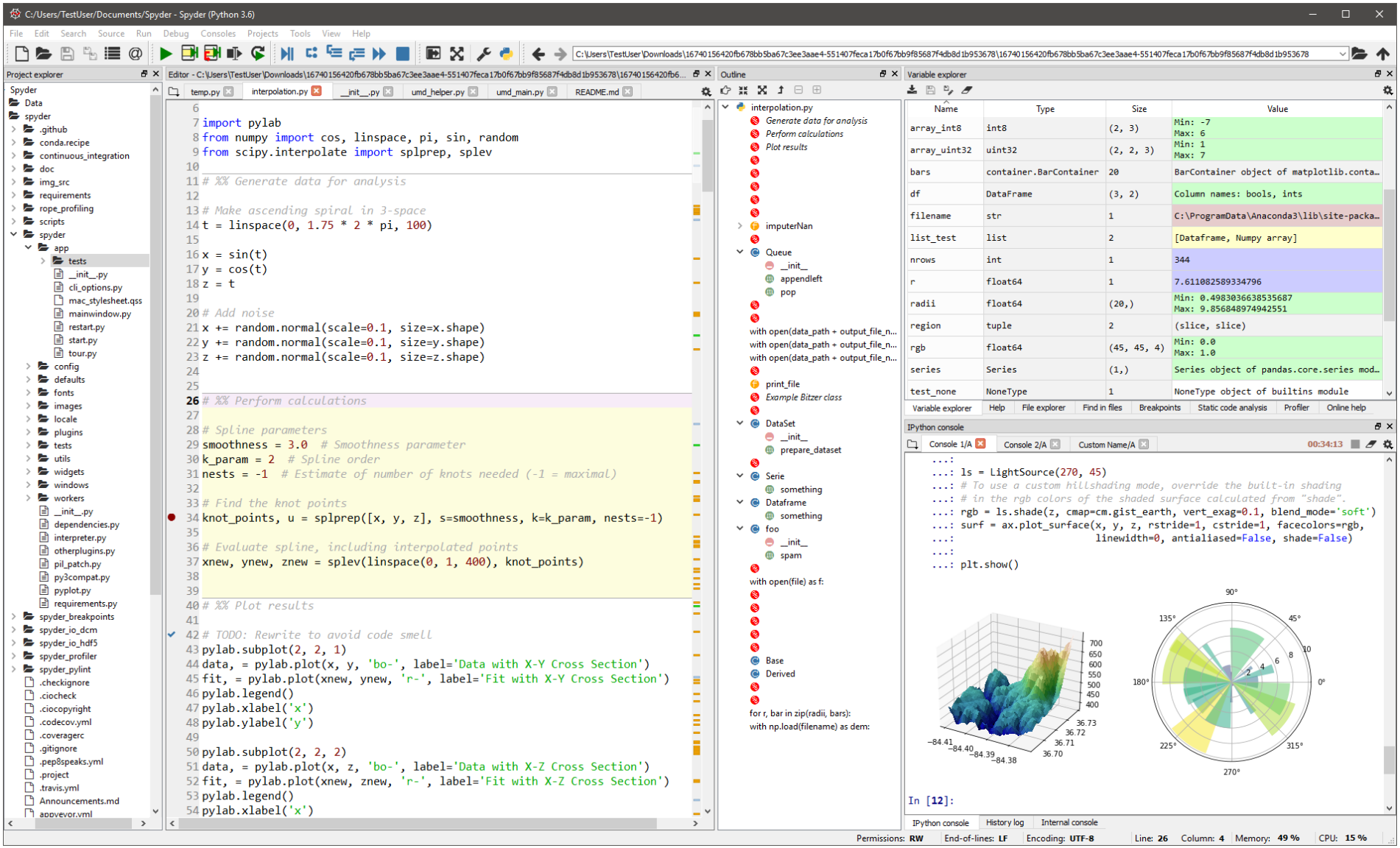
Data	
dat	33 obs. of 11 variables
Values	
M	List of 3
url	"http://www.gdacs.org/Cyclones/report.aspx?eventid=41058&e…

Files | Plots | Packages | Help | Viewer



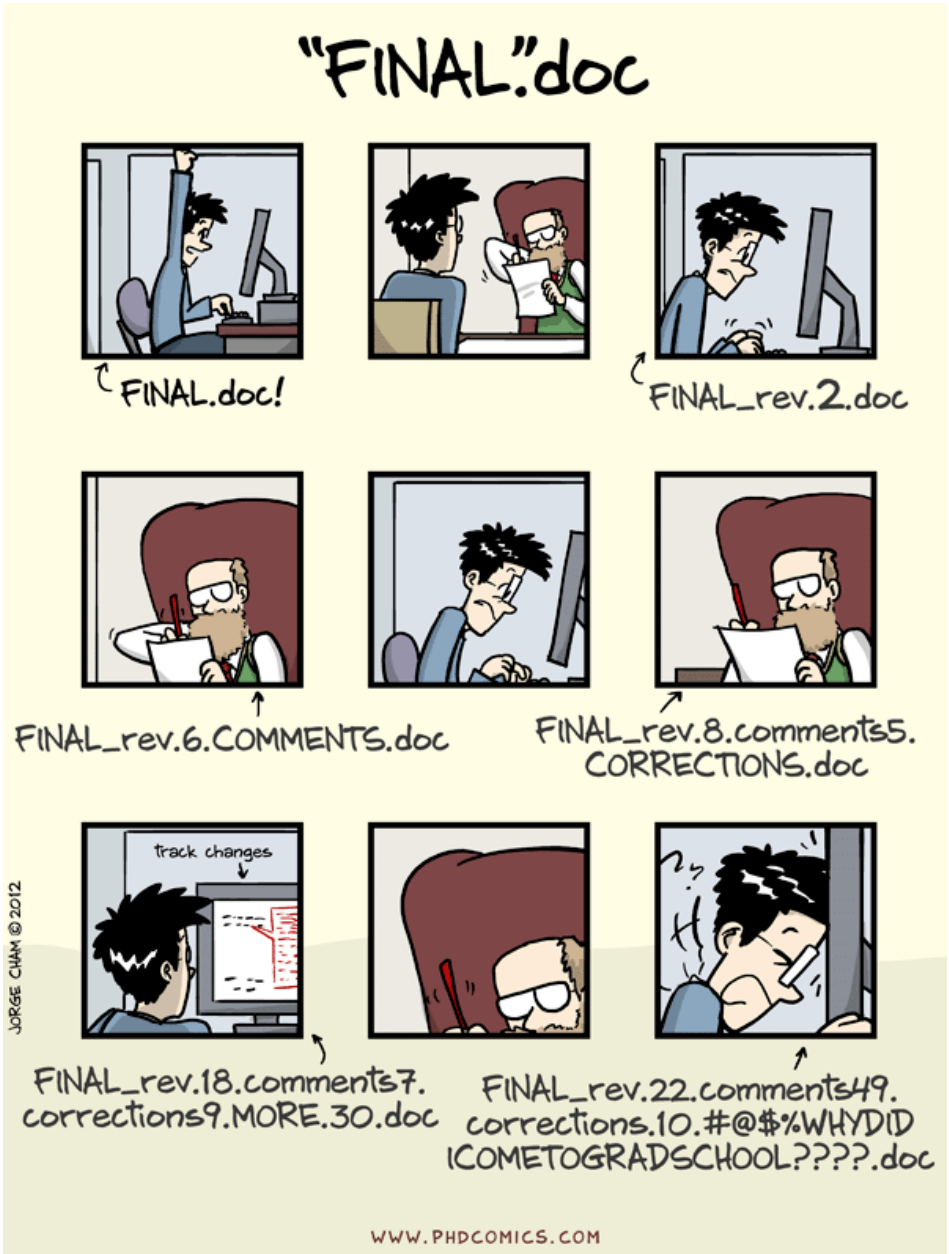
Data: dat • Chart ID: GeoChartID926a2a743168
R version 3.0.2 (2013-09-25) • googleVis-0.4.7 • Google Terms of Use • Data Policy

Spyder

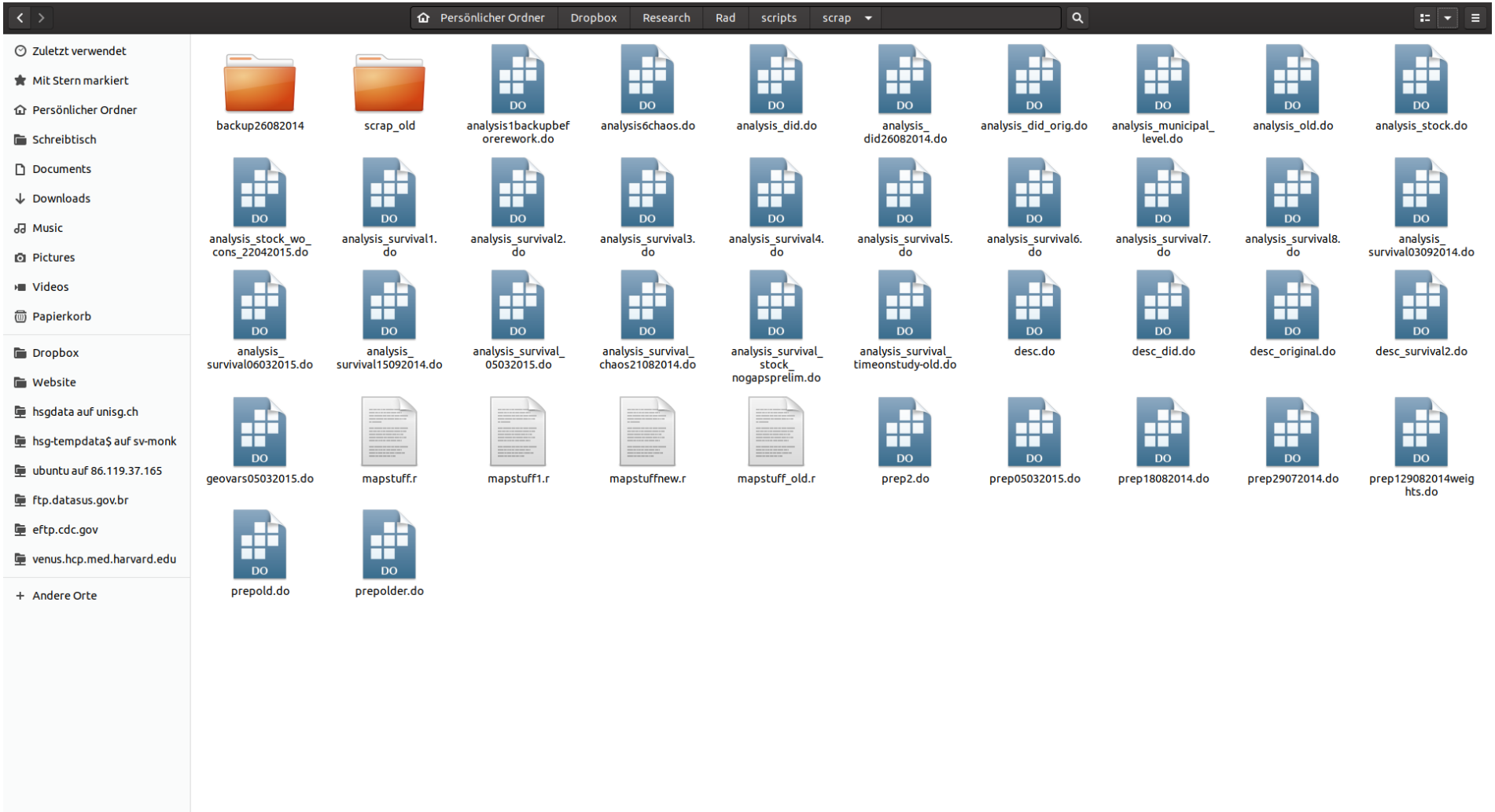

C:/Users/TestUser/Documents/Spyder - Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Project explorer
Editor - C:\Users\TestUser\Downloads\16740156420fb678bb5ba67c3ee3aae4-551407feca17b0f67bb9f85687f4db8d1b953678\16740156420fb6...
temp.py
interpolation.py
__init__.py
umd_helper.py
umd_main.py
README.md
import pylab
from numpy import cos, linspace, pi, sin, random
from scipy.interpolate import splprep, splev
# %% Generate data for analysis
# Make ascending spiral in 3-space
t = linspace(0, 1.75 * 2 * pi, 100)
x = sin(t)
y = cos(t)
z = t
# Add noise
x += random.normal(scale=0.1, size=x.shape)
y += random.normal(scale=0.1, size=y.shape)
z += random.normal(scale=0.1, size=z.shape)
# %% Perform calculations
# Spline parameters
smoothness = 3.0 # Smoothness parameter
k_param = 2 # Spline order
nests = -1 # Estimate of number of knots needed (-1 = maximal)
# Find the knot points
knot_points, u = splprep([x, y, z], s=smoothness, k=k_param, nests=-1)
Evaluate spline, including interpolated points
xnew, ynew, znew = splev(linspace(0, 1, 400), knot_points)
# %% Plot results
# TODO: Rewrite to avoid code smell
pylab.subplot(2, 2, 1)
data, = pylab.plot(x, y, 'bo-', label='Data with X-Y Cross Section')
fit, = pylab.plot(xnew, ynew, 'r-', label='Fit with X-Y Cross Section')
pylab.legend()
pylab.xlabel('x')
pylab.ylabel('y')
pylab.subplot(2, 2, 2)
data, = pylab.plot(x, z, 'bo-', label='Data with X-Z Cross Section')
fit, = pylab.plot(xnew, znew, 'r-', label='Fit with X-Z Cross Section')
pylab.legend()
pylab.xlabel('x')
Outline
Variable explorer
Name Type Size Value
array_int8 int8 (2, 3) Min: -7 Max: 6
array_uint32 uint32 (2, 2, 3) Min: 1 Max: 7
bars container.BarContainer 20 BarContainer object of matplotlib.conta...
df DataFrame (3, 2) Column names: bools, ints
filename str 1 C:\ProgramData\Anaconda3\lib\site-packa...
list_test list 2 [Dataframe, Numpy array]
nrows int 1 344
r float64 1 7.611082589334796
radii float64 (20,) Min: 0.4983036638535687 Max: 9.856848974942551
region tuple 2 (slice, slice)
rgb float64 (45, 45, 4) Min: 0.0 Max: 1.0
series Series (1,) Series object of pandas.core.series mod...
test_none NoneType 1 NoneType object of builtins module
Variable explorer Help File explorer Find in files Breakpoints Static code analysis Profiler Online help
IPython console
Console 1/A Console 2/A Custom Name/A 00:34:13
...: ls = LightSource(270, 45)
...: # To use a custom hillshading mode, override the built-in shading
...: # in the rgb colors of the shaded surface calculated from "shade".
...: rgb = ls.shade(z, cmap=cm.gist_earth, vert_exag=0.1, blend_mode='soft')
...: surf = ax.plot_surface(x, y, z, rstride=1, cstride=1, facecolors=rgb,
...:   linewidth=0, antialiased=False, shade=False)
...: plt.show()
In [12]:
IPython console History log Internal console
Permissions: RW End-of-lines: LF Encoding: UTF-8 Line: 26 Column: 4 Memory: 49 % CPU: 15 %

Version control

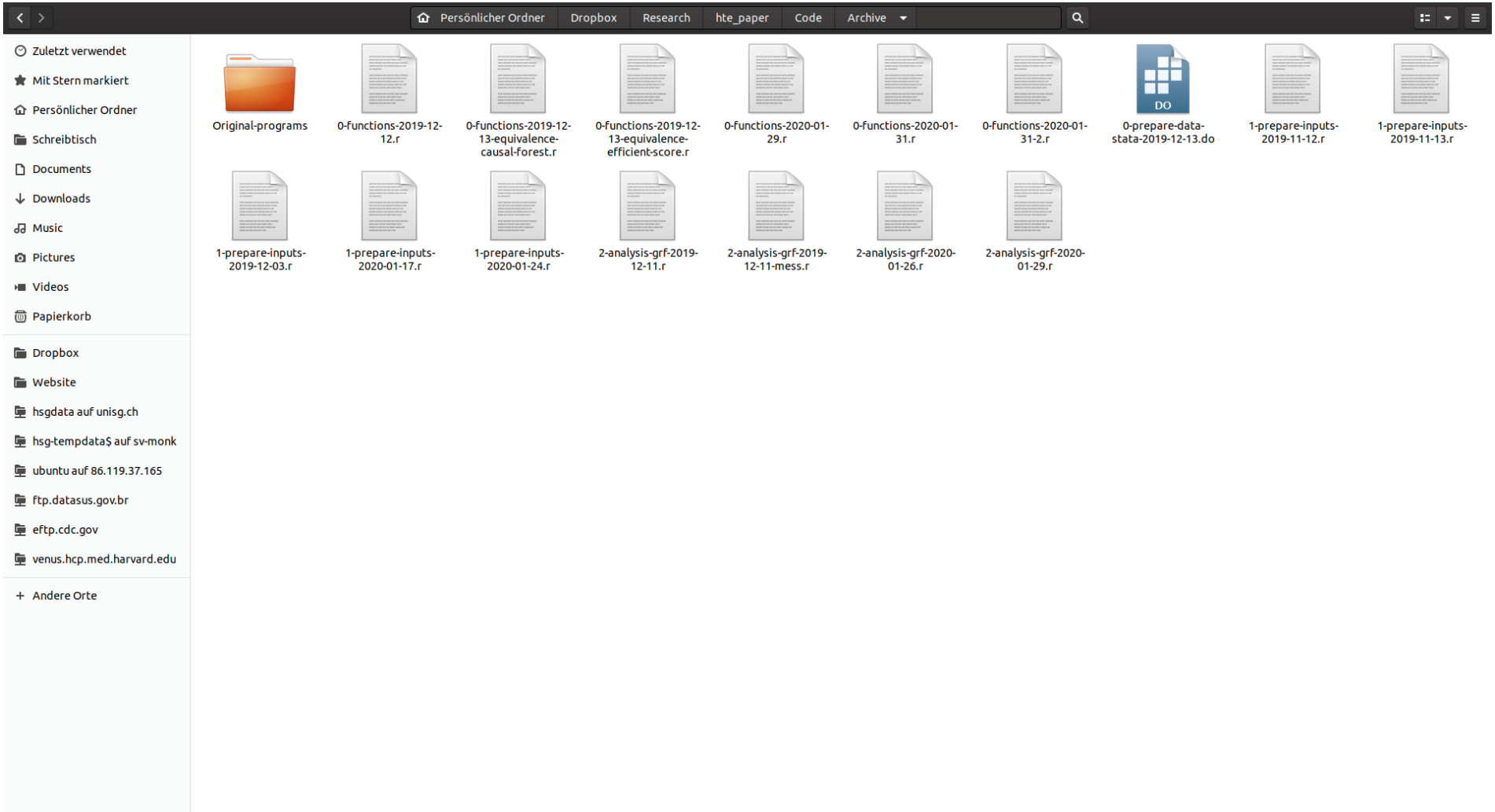
Familiar?


"FINAL".doc
FINAL.doc!
FINAL_rev.2.doc
FINAL_rev.6.COMMENTS.doc
FINAL_rev.8.comments5.
CORRECTIONS.doc
track changes
FINAL_rev.18.comments7.
corrections9.MORE.30.doc
FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc
JORGE CHAM © 2012
WWW.PHDCOMICS.COM

Familiar?


Persönlicher Ordner
Dropbox
Research
Rad
scripts
scrap
Zuletzt verwendet
Mit Stern markiert
Persönlicher Ordner
Schreibtisch
Documents
Downloads
Music
Pictures
Videos
Papierkorb
Dropbox
Website
hsgdata auf unisg.ch
hsg-tempdata$ auf sv-monk
ubuntu auf 86.119.37.165
ftp.datasus.gov.br
eftp.cdc.gov
venus.hcp.med.harvard.edu
Andere Orte
backup26082014
scrap_old
analysis1backupbeforerework.do
analysis6chaos.do
analysis_did.do
analysis_did26082014.do
analysis_did_orig.do
analysis_municipal_level.do
analysis_old.do
analysis_stock.do
analysis_stock_wo_cons_22042015.do
analysis_survival1.do
analysis_survival2.do
analysis_survival3.do
analysis_survival4.do
analysis_survival5.do
analysis_survival6.do
analysis_survival7.do
analysis_survival8.do
analysis_survival03092014.do
analysis_survival06032015.do
analysis_survival15092014.do
analysis_survival_05032015.do
analysis_survival_chaos21082014.do
analysis_survival_stock_nogapsprelim.do
analysis_survival_timeonstudy-old.do
desc.do
desc_did.do
desc_original.do
desc_survival2.do
geovars05032015.do
mapstuff.r
mapstuff1.r
mapstuffnew.r
mapstuff_old.r
prep2.do
prep05032015.do
prep18082014.do
prep29072014.do
prep129082014weights.do
prepold.do
prepolder.do

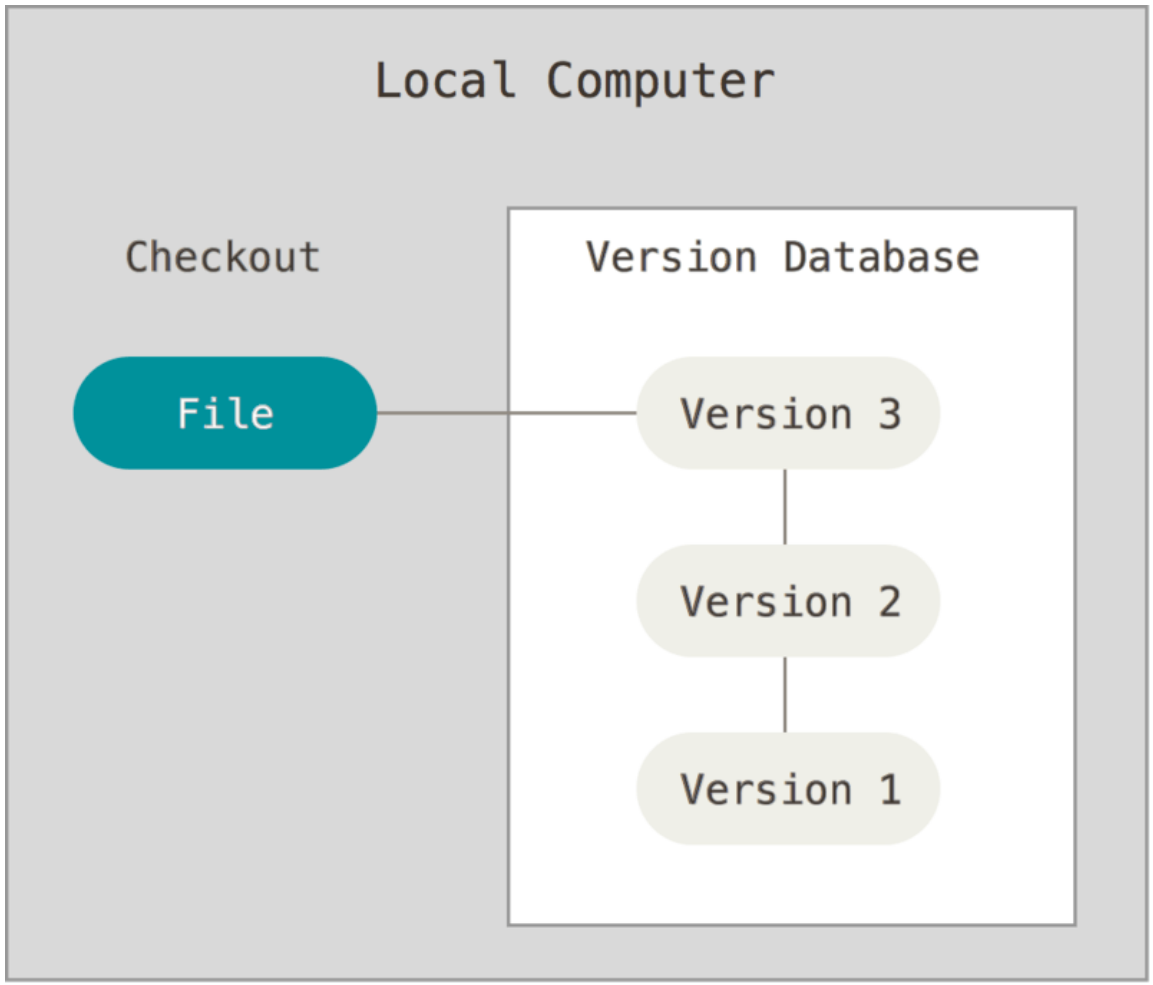
Familiar?


Persönlicher Ordner
Dropbox
Research
hte_paper
Code
Archive
Zuletzt verwendet
Mit Stern markiert
Persönlicher Ordner
Schreibtisch
Documents
Downloads
Music
Pictures
Videos
Papierkorb
Dropbox
Website
hsgdata auf unisg.ch
hsg-tempdata$ auf sv-monk
ubuntu auf 86.119.37.165
ftp.datasus.gov.br
eftp.cdc.gov
venus.hcp.med.harvard.edu
Andere Orte
Original-programs
0-functions-2019-12-12.r
0-functions-2019-12-13-equivalence-causal-forest.r
0-functions-2019-12-13-equivalence-efficient-score.r
0-functions-2020-01-29.r
0-functions-2020-01-31.r
0-functions-2020-01-31-2.r
0-prepare-data-stata-2019-12-13.do
1-prepare-inputs-2019-11-12.r
1-prepare-inputs-2019-11-13.r
1-prepare-inputs-2019-12-03.r
1-prepare-inputs-2020-01-17.r
1-prepare-inputs-2020-01-24.r
2-analysis-grf-2019-12-11.r
2-analysis-grf-2019-12-11-mess.r
2-analysis-grf-2020-01-26.r
2-analysis-grf-2020-01-29.r

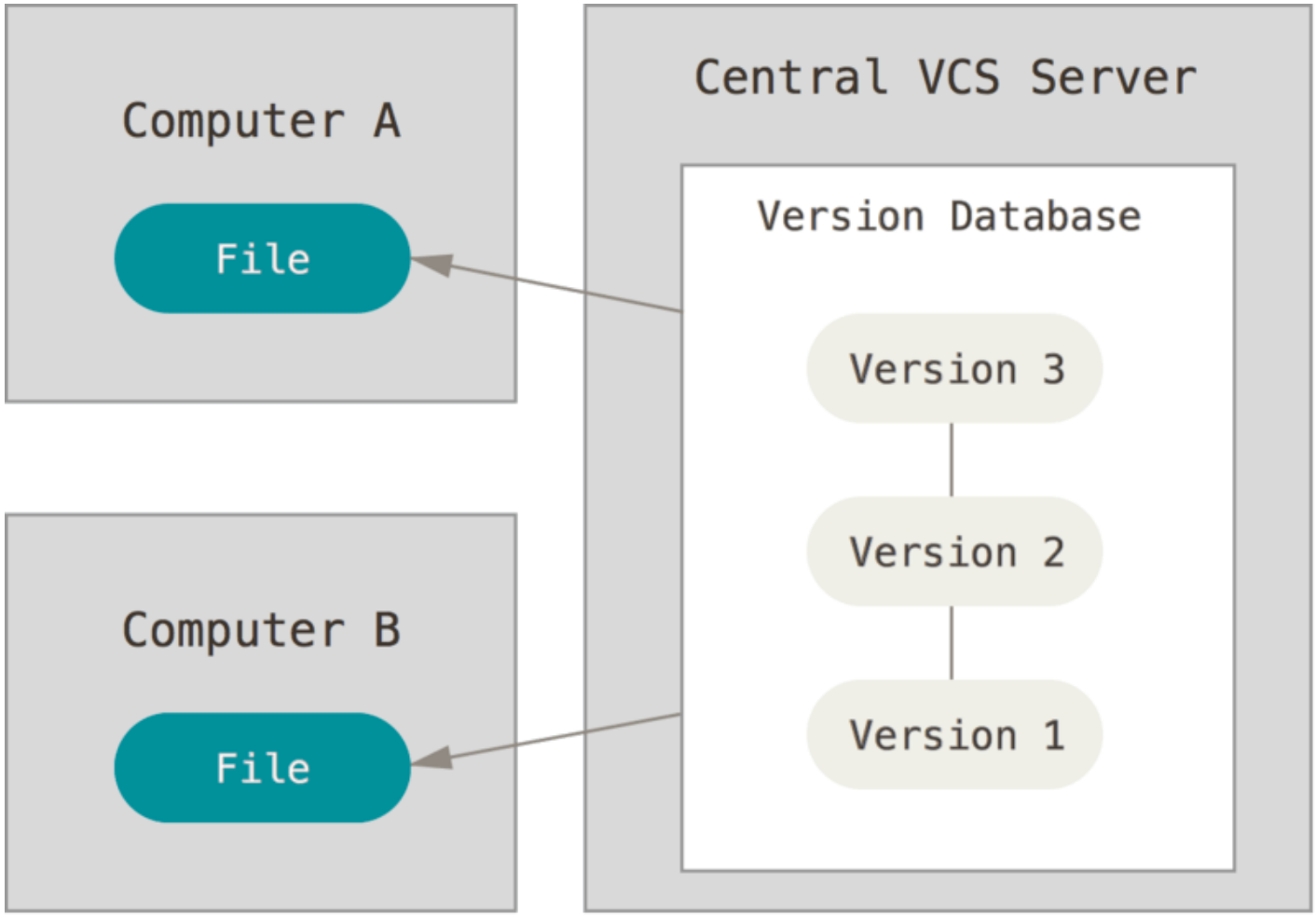
Tracking different versions

- Different options:
  - Don’t keep track.
  - Keep file copies/zip archives. Adhere to a versioning format. Diff files.
  - Formal version control.
- Formal version control:
  - Ubiquitous in software development, extremely useful for many purposes.
  - Less established for statistical data management and analysis.
  - Workflow in data analysis differs. More effort to use properly.
  - Viewpoints on usefulness for data analysis vary.

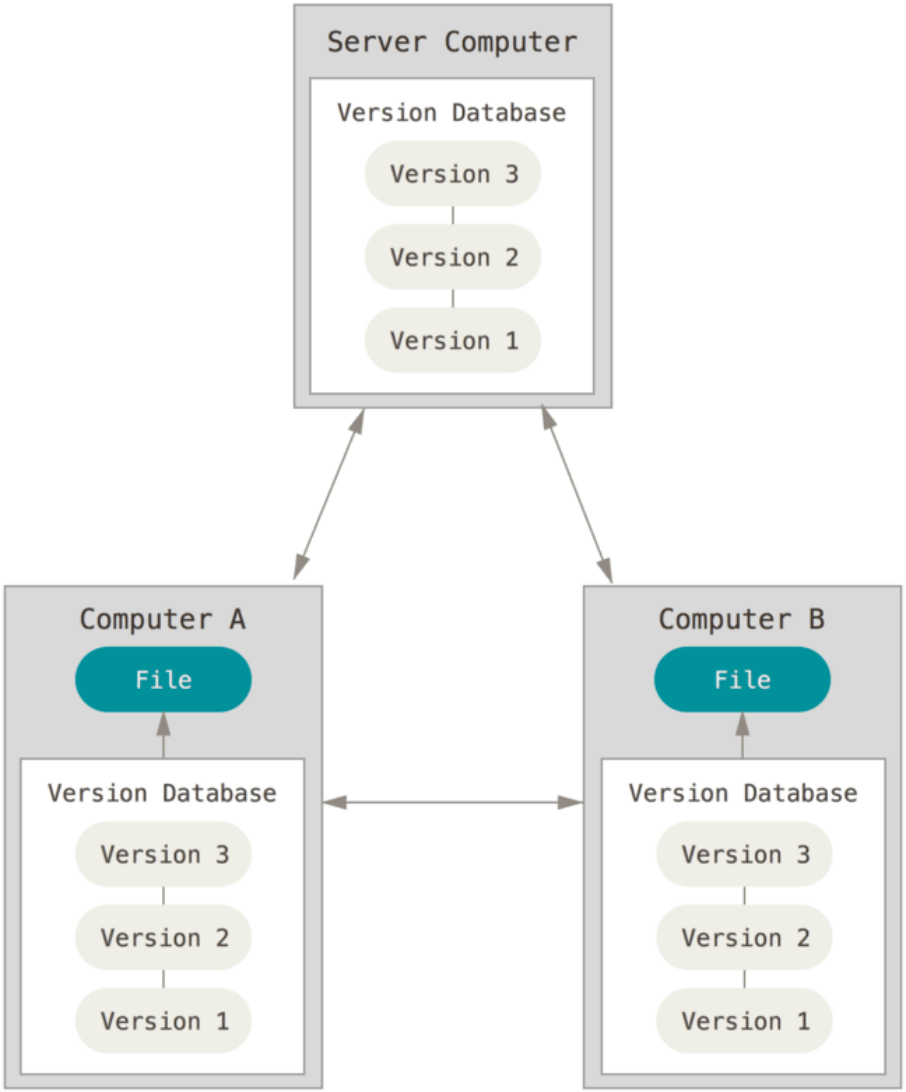
Local version control



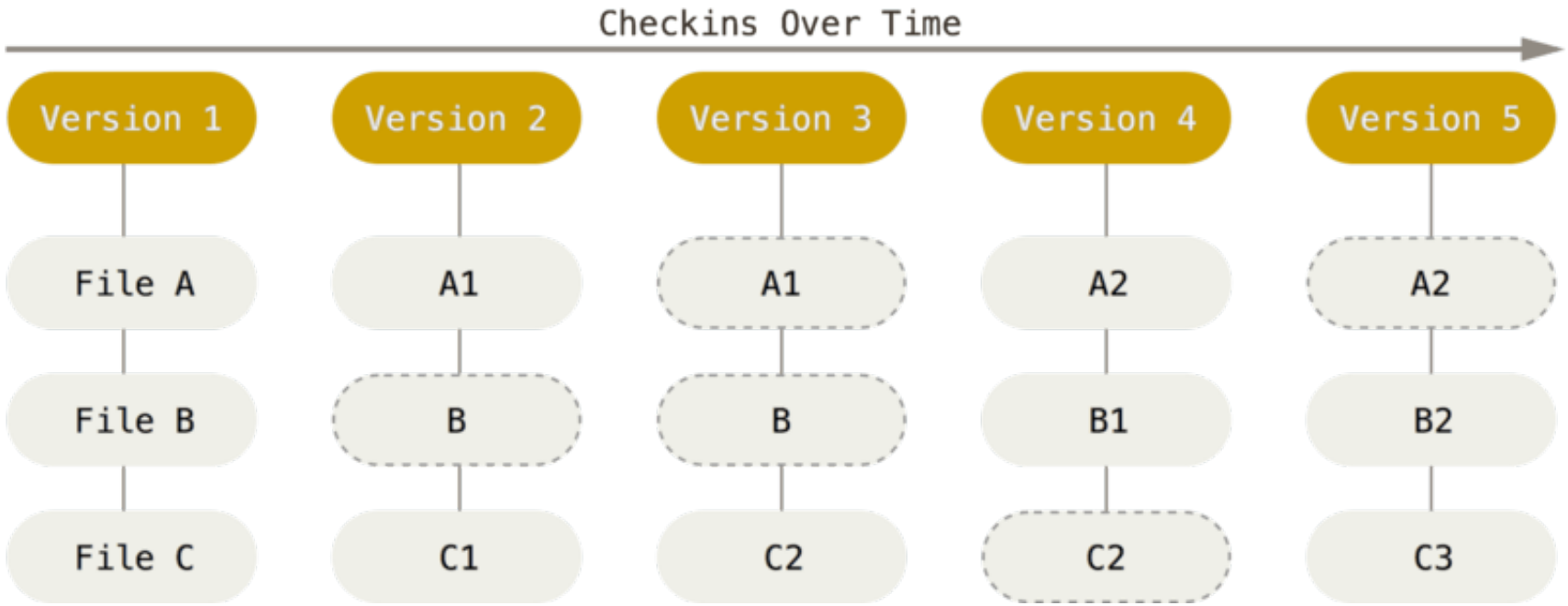
Centralized version control



Distributed version control



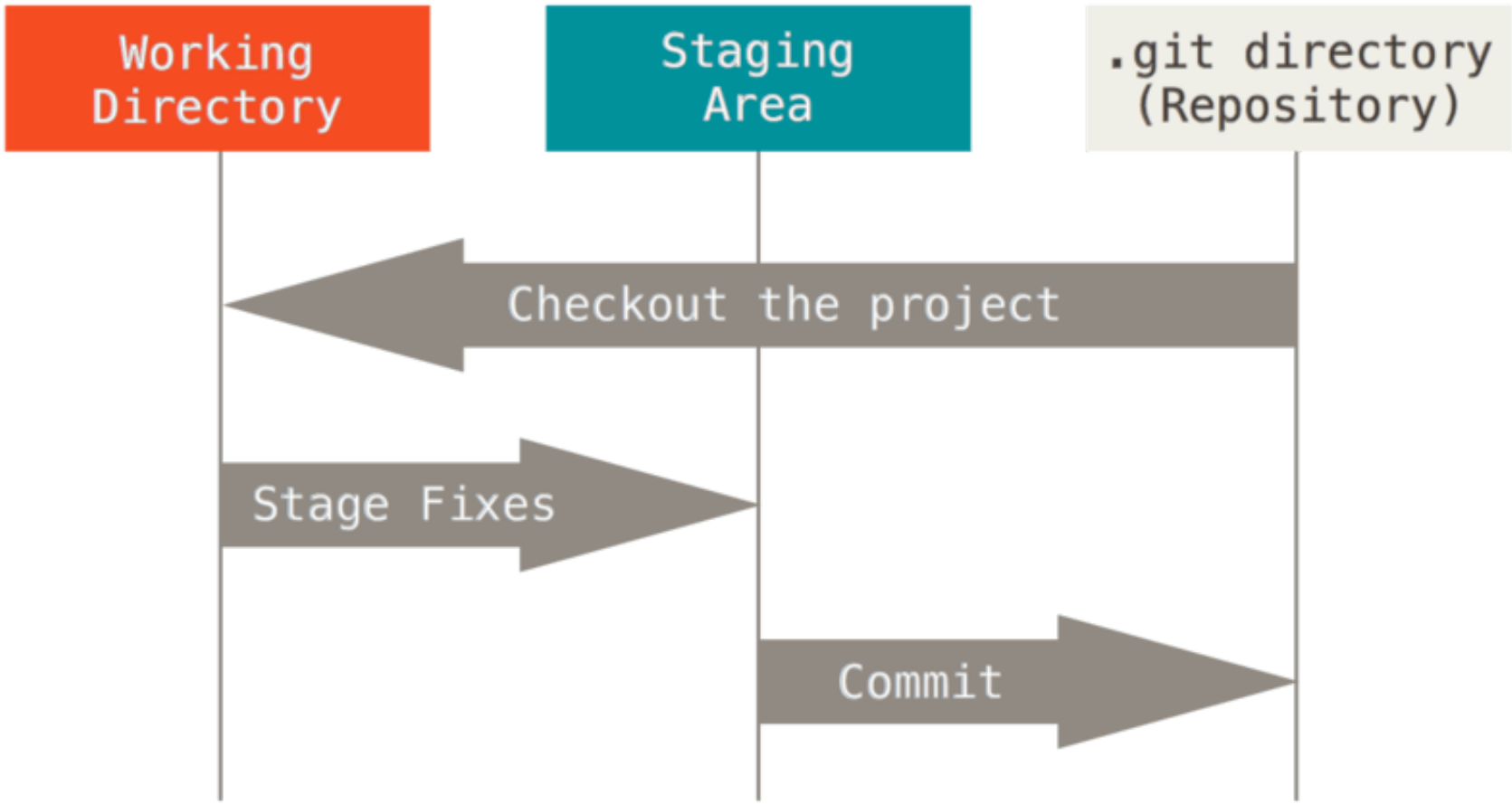
Distributed version control - snapshots



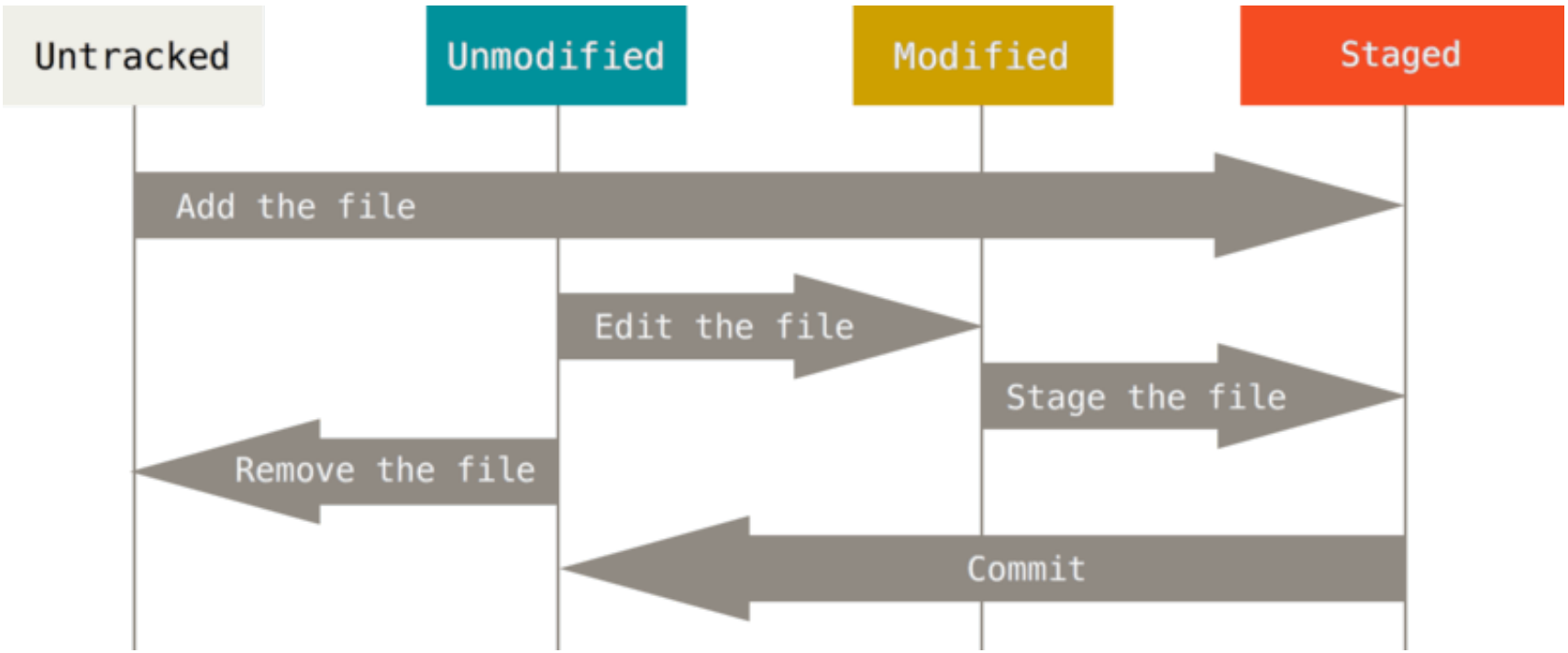
Git

- Git is the dominant *distributed* version control system today.
- Developed by Linus Torvalds, the creator of the Linux kernel.
- Can be used on the command line. GUIs/editor plugins available. Integrated into RStudio.
- Fast, good support for non-linear development (thousands of parallel branches).
- Tracks any content (but mostly plain text files).
  - Source code, manuscripts, websites, presentations, ...
  - As a rule, do not version control data.

Workflow: Areas



Workflow: File lifecycle



Git

- Git stores the complete history of files and all changes you or others made to them in a local repository on your computer.
- You can have different ‘branches’ with different features, switch between them or merge them.
- You can keep a ‘remote’ copy of the repository on a server, but this is not required.
- You can ‘clone’ repositories from others and ‘push’ back changes you made to them.
- Public repositories facilitate collaboration and development.

Advantages and disadvantages

- Easy to track changes over time (from multiple people) and simple to revert them.
- Helps structure thoughts and makes you write cleaner code.
- Reproducible. Eases outside contributions and helps hunting down bugs.

- A lot of work (to learn and to use), especially in the short term.
- More so if you have to convince your colleagues.
- Integrates less well with iterative back-and-forth workflow.

Advantages and disadvantages

- Easy to track changes over time (from multiple people) and simple to revert them.
- Helps structure thoughts and makes you write cleaner code.
- Reproducible. Eases outside contributions and helps hunting down bugs.

- A lot of work (to learn and to use), especially in the short term.
- More so if you have to convince your colleagues.
- Integrates less well with iterative back-and-forth workflow.

➥ For many projects

➥ Still good to understand the basics, you are likely to need them. Lots of programs are hosted on public git repositories, firms use it.

Resources

- ProGit is a good and free resource. Skim the first few chapters.
- Software carpentry has a good introductory course. Many more resources online.
- Github is a site for online storage of Git repositories (Git $\neq$ Github!). You can create a remote repository there and push code to it. Public repositories are free.
- Github also offers student and educator accounts including free private repositories.
- Some editors offer *persistent undo*, which can be very helpful (as long as you are aware of its limits—it is not a replacement for version control).

Command line interface

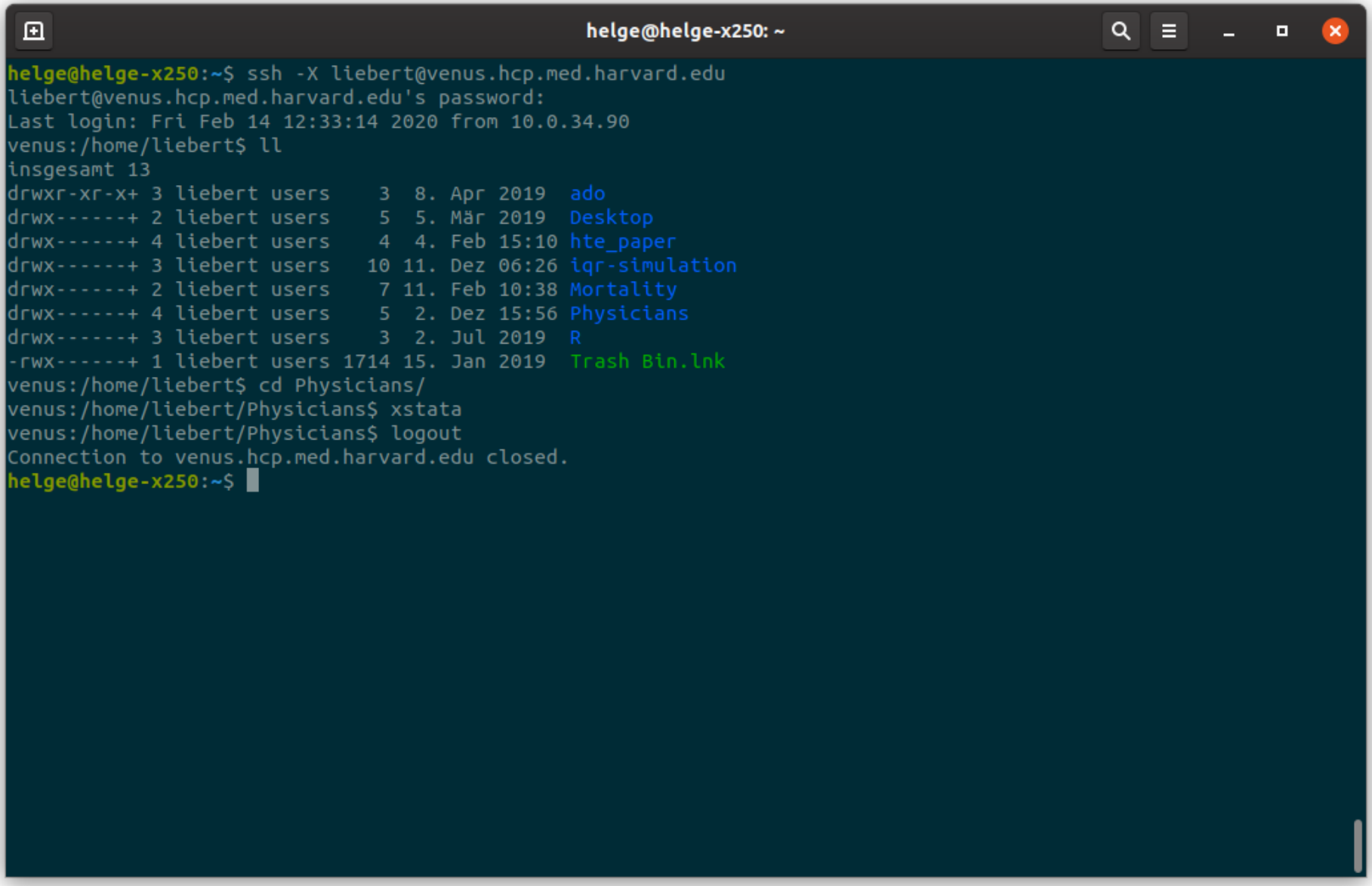
Command line interfaces

- A *command-line interface* (CLI) processes commands to a computer program in the form of lines of text.
- Different from *graphical user interfaces* (GUIs).
- The program which handles the interface is called a *command-line interpreter* or *shell*.
- A shell provides interactive or programmatic access to operating system functions and other services.
- CLI shells became the primary access to computer terminals starting in the mid-1960s. Also used by mainstream personal computers (Unix, MS-DOS, ...).

Terminals


SLMIN .SYS 12P 20-Dec-85
XL .SYS 4P 20-Dec-85
SP .SYS 6P 13-Aug-86
RT11SJ.SYS 70P 13-Aug-86
HL .SYS 2P 13-Aug-86
SB .SYS 5P 31-May-85
BASIC .SAV 56 24-May-79
DATIME.SAV 4 20-Dec-85
DUMP .SAV 9 20-Dec-85
TSXMOD.SAV 78 27-Nov-92
HARRIS.SAV 41 12-Jun-85
START .P36 2 21-Dec-91
EM1A .STN 19P 02-Feb-93
STAND .LIN 12P 15-Aug-83
EM1B .STN 19 11-Feb-93
JKFLIP.FOR 3 08-Mar-96
ANNOT .SAV 38 18-Apr-93
DUO .DIR 18 16-Jul-96
DIR .DIF 26 16-Jul-96
DEMOBC.OBJ 1 -BAD-
114 Files, 5949 Blocks
14433 Free blocks
.o
VM .SYS 3P 13-Aug-86
LD .SYS 8P 23-Aug-86
BL .SYS 5P 13-Aug-86
BU .SYS 8P 13-Aug-86
TT .SYS 2P 13-Aug-86
KL02DC.SYS 71P 21-Nov-84
BINCON.SAV 24 20-Dec-85
DIR .SAV 19 20-Dec-85
DUP .SAV 47 20-Dec-85
FORTRA.SAV 206 21-Aug-85
LET .SAV 5 20-Dec-85
METRO .OBJ 1519P 16-Aug-88
TSXUCL.TSX 22 07-Mar-92
TSXY6 .MSG 1P 04-Sep-85
JKFLIP.SAV 30 08-Mar-96
RT11FB.SYS 93P 20-Dec-85
TSXP23.NEW 1200P 27-Nov-92
DL .DIR 7 16-Jul-96
DEMOFG.OBJ 1 -BAD-
EVAN .ID 1
digital VT100

Terminals


helge@helge-x250: ~
helge@helge-x250:~$ ssh -X liebert@venus.hcp.med.harvard.edu
liebert@venus.hcp.med.harvard.edu's password:
Last login: Fri Feb 14 12:33:14 2020 from 10.0.34.90
venus:/home/liebert$ ll
insgesamt 13
drwxr-xr-x+ 3 liebert users   3 8. Apr 2019 ado
drwx------+ 2 liebert users   5 5. Mär 2019 Desktop
drwx------+ 4 liebert users   4 4. Feb 15:10 hte_paper
drwx------+ 3 liebert users   10 11. Dez 06:26 iqr-simulation
drwx------+ 2 liebert users   7 11. Feb 10:38 Mortality
drwx------+ 4 liebert users   5 2. Dez 15:56 Physicians
drwx------+ 3 liebert users   3 2. Jul 2019 R
-rwx------+ 1 liebert users 1714 15. Jan 2019 Trash Bin.lnk
venus:/home/liebert$ cd Physicians/
venus:/home/liebert/Physicians$ xstata
venus:/home/liebert/Physicians$ logout
Connection to venus.hcp.med.harvard.edu closed.
helge@helge-x250:~$

Operating system shells

- Still in use today. Unix shells most relevant.
- Unix shells are used in Unix-derived systems (Linux, MacOS).
- Common operating system shells: `sh` (Unix), `cmd` (Windows); `bash`, `zsh` (Linux, MacOS), ...
- Unix-based operating systems (Linux, MacOS) have Bash/zsh pre-installed. Just start `Terminal`.
- On Linux, all system maintenance and software installation can be done via the shell.

Why does this matter?

- In 2019, 100% of the world's TOP500 supercomputers run on Linux.
- 96.3% of the world's top 1 million servers run on Linux.
- 90% of all cloud infrastructure operates on Linux.

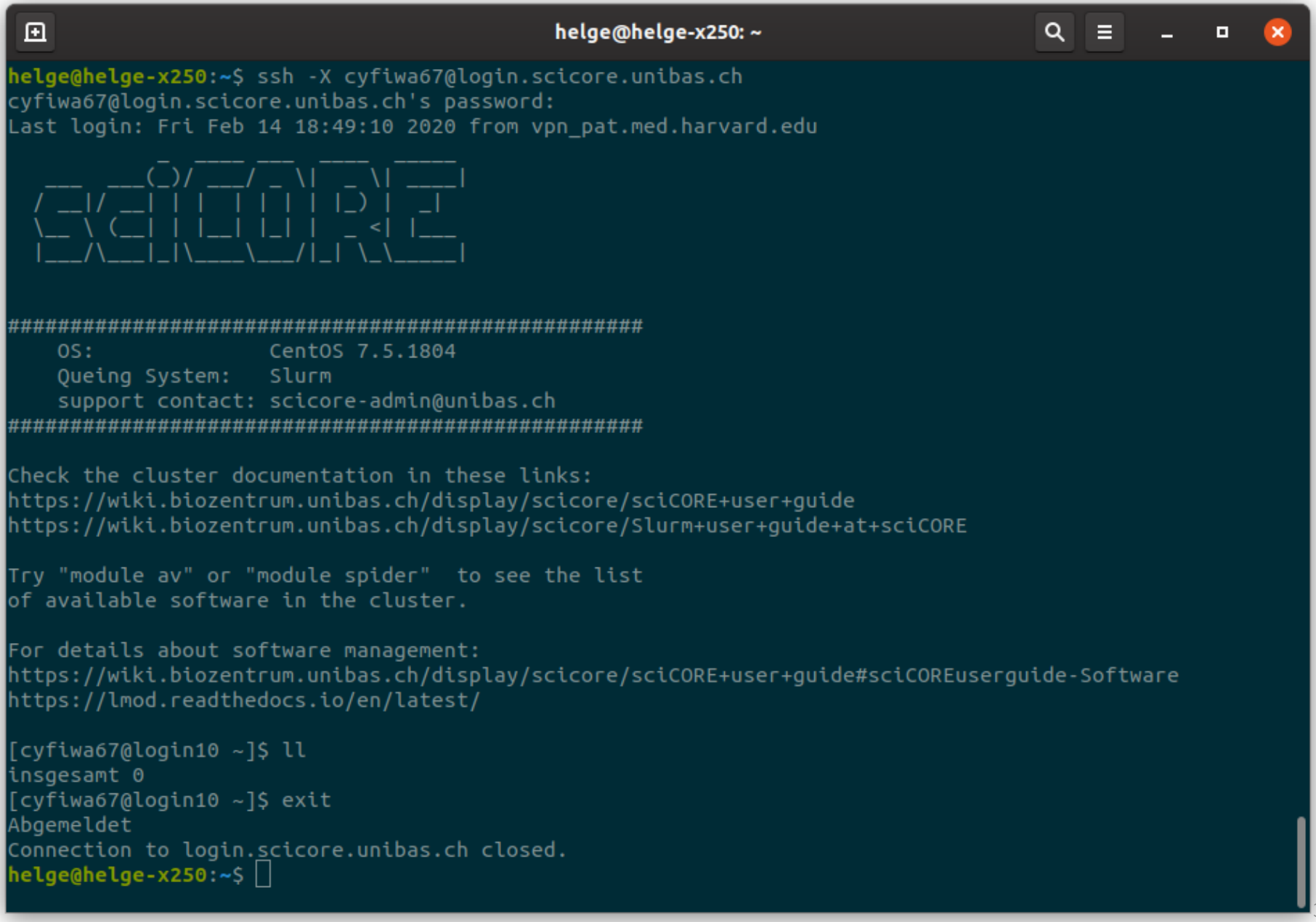
Why does this matter?

- In 2019, 100% of the world's TOP500 supercomputers run on Linux.
- 96.3% of the world's top 1 million servers run on Linux.
- 90% of all cloud infrastructure operates on Linux.

- sciCORE runs on Linux and uses CLI access.
- Switch engines provides Linux instances with CLI access.
- Amazon web services (AWS) provide Linux-based computing resources.
- So do Swisscom dynamic computing services.

➥ Much of the computing infrastructure you are likely to use will run Linux.

sciCORE


helge@helge-x250: ~
helge@helge-x250:~$ ssh -X cyfiwa67@login.scicore.unibas.ch
cyfiwa67@login.scicore.unibas.ch's password:
Last login: Fri Feb 14 18:49:10 2020 from vpn_pat.med.harvard.edu
OS: CentOS 7.5.1804
Queing System: Slurm
support contact: scicore-admin@unibas.ch
Check the cluster documentation in these links:
https://wiki.biozentrum.unibas.ch/display/scicore/sciCORE+user+guide
https://wiki.biozentrum.unibas.ch/display/scicore/Slurm+user+guide+at+sciCORE
Try "module av" or "module spider" to see the list
of available software in the cluster.
For details about software management:
https://wiki.biozentrum.unibas.ch/display/scicore/sciCORE+user+guide#sciCOREuserguide-Software
https://lmod.readthedocs.io/en/latest/
[cyfiwa67@login10 ~]$ ll
insgesamt 0
[cyfiwa67@login10 ~]$ exit
Abgemeldet
Connection to login.scicore.unibas.ch closed.
helge@helge-x250:~$

A note on operating systems

- MacOS or Linux offer built-in access to a Unix shell (Bash or zsh).
- Further software is managed via a package management system and distributed via software repositories.
- On Linux, use your package manager to install anything you require.
- On MacOS, familiarize yourself with Homebrew. Install iterm2 if you want a fancier terminal.

A note on operating systems

- MacOS or Linux offer built-in access to a Unix shell (Bash or zsh).
- Further software is managed via a package management system and distributed via software repositories.
- On Linux, use your package manager to install anything you require.
- On MacOS, familiarize yourself with Homebrew. Install iterm2 if you want a fancier terminal.

- Windows does not provide proper access to a Unix shell out-of-the-box.
- Many tools are not available or cumbersome to use.
- Windows has cmd (or PowerShell). These are not Unix shells and not a viable replacement for most use cases.
- Dependency resolution can be a nightmare. Package managers for Windows like scoop (or chocolatey) help.

What is a package manager?

> *"A package manager or package-management system is a collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner."*

Why use it?

- Things work. Dependencies are automatically resolved.
- System and software is kept up to date.
- Bugfixes and (critical) patches are distributed automatically.
- Minimum of active maintenance.

On Windows

- Use scoop or chocolatey to manage software on your computer.
- The most popular version control (Git) is distributed with a slimmed-down version of bash on Windows (Git Bash). So is Anaconda, a popular Python distribution.
- In recent versions of Windows, Microsoft has implemented a Linux compatibility layer, Windows Subsystem for Linux (WSL).
- If you are on Windows, WSL, MSYS2, Cygwin or Git Bash *may* work as a replacement.
- Often, all you need is to connect to a server via `ssh`. Use MobaXterm or putty. *The server provides the shell environment.*

Simple bash usage

- *Return* executes a command, *CTRL + c* aborts a process, *CTRL + d* quits.
- Multiple commands can be run as scripts (suffix `.sh`).
- Comment character is #.
- Some examples:

```
cd somedir/subdir # Navigate to a folder, (c)hange (d)irectory
cd .. # Navigate to parent directory
cd # Return to your home folder
ls # List directory contents
mv file subdir/file # Move a file
cp file ../file # Copy a file
rm myscript.ch # Delete a file
R # Start the R console
man ssh # Display manual pages for the ssh program
shutdown -h +20 # Shut off computer in 20 minutes
```

Resources

- Basic knowledge often sufficient. Examples cover some ground.
- Google and O'Reilly/No Starch books and pocket references are handy.
- Software carpentry has an introductory course.
- Advanced Bash-Scripting Guide

More examples

```
# Secure shell login to remote server
ssh myusername@13.438.14.673

# Edit your R script with vim
vim myscript.r
# Search for file and open it
vim $(find -name somefile.txt)

# Execute your R script
R -f myscript.r
# Execute your Python script
python myscript.py
# Execute a shell script
sh myscript.sh

# Stage file for version control
git add myscript.py
# Commit file to version control tree
git commit myscript.py
```

Searching and filtering

- Great capabilities for pattern matching.

```
# Rename files by pattern, e.g. 'data_file1.csv' -> 'file1.csv'
rename data_ '' data_*.csv
```

- Searching/filtering of text is a very popular application.
- Popular unix filters
  - cat, head, tail, tee, wc, ...
  - cut, paste
  - find
  - sort, uniq
  - diff, comm, cmp
  - grep (or ag, ripgrep), sed, awk
  - (perl, python, ...)

Searching and filtering

- Sounds tedious? It can be. But it can also be extremely powerful.
- `grep` is a popular command line utility for searching plain text files.

```
# Search for keyword in filename
grep "keyword" filename
# Search for keyword, (i)gnore case, (r)ecursively, all files
grep -ir "keyword" *
# Print matching line and line (n)umber to output-file
grep -n "keyword" *.txt > output-file
```

Search patterns

- Convert all pdf files in a folder to text and search them.

```
for file in *.pdf; do pdftotext "$file"; done
grep -ir "keyword" *.txt
```

- Searching can be more complex than just keywords.
- Suppose you need to find all documents with sentences expressing proficiency in Spanish or Italian, but not English.

```
grep -ir
"(fluent|proficient)(?!.*?english)[^\.]*?(spanish|italian).*?\."
*.txt
```

Regular expressions

Regular expressions

- `grep`: **g**lobally search a **r**egular **e**xpression and **p**rint.
- A regular expression is a pattern that describes a set of strings.
- Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.
- Usually used for find/replace operations on strings, or for validation.
- Pervasive in Unix text processing programs (`grep` was originally written by Ken Thompson).
- Not limited to `grep`!

Regular expressions

- *Pattern matching*: Find one of a specified set of strings in text.
- Examples:
  - Diagnoses in medical records.
  - Addresses or zip codes in concatenated admin records.
  - Sequences within a genome, e.g. a virus signature.
  - Validate data-entry fields (URL, date, email, credit card #).
  - Example using a regex tester.

Examples

AHVN 13: `756\.[0-9]{4}\.[0-9]{4}\.[0-9]{4}\.[0-9]{2}`
Matches: `756.1234.5678.90`
Does not match: `123.45.678.675`

US-SSN: `[0-9]{3}-[0-9]{2}-[0-9]{4}`
Matches: `166-11-4433`
Does not match: `11-55555555`

Email addresses: `[a-z]+@([a-z]+\.)+(ch|edu|com)`
Matches: `someone@unibas.ch`
Does not match: `someone@invalid.domain`

Screening job candidates

```
" [First name]! and pre/2 [last name] w/7
   bush or gore or republican! or democrat! or charg!
   or accus! or criticiz! or blam! or defend! or iran contra
   or clinton or spotted owl or florida recount or sex!
   or controvers! or fraud! or investigat! or bankrupt!
   or layoff! or downsiz! or PNTR or NAFTA or outsourc!
   or indict! or enron or kerry or iraq or wmd! or arrest!
   or intox! or fired or racis! or intox! or slur!
   or controvers! or abortion! or gay! or homosexual!
   or gun! or firearm! "
```

— LexisNexis search string used by Monica Goodling to illegally screen candidates for DOJ positions

LexisNexis™

http://www.justice.gov/oig/special/s0807/final.pdf

Regular expressions

- Characters in a regular expression are either regular characters (literal meaning) or metacharacters (special meaning).
- Generally, letters and numbers match themselves.
- Normally case sensitive, but can be set to ignore case.
- Careful with puntuation, most of it has special meanings.
- To match metacharacters literally, they need to be *escaped*, i.e. preceded by a backslash `\`.

Matching string literals

Regular expression	Input string
`input`	`This input string is short.`
`15`	`The due date is 15.12.`

Matching string literals

Regular expression	Input string
`input`	`This input string is short.`
`15`	`The due date is 15.12.`
`15.12`	`The due date is 15.12.`

Matching string literals

Regular expression	Input string
`input`	`This input string is short.`
`15`	`The due date is 15.12.`
`15.12`	`The due date is 15.12.`

but:

`15.12`	`The due date is 15712.`

match `.` literal:

`15\.12`	`The due date is 15712.`
`15\.12`	`The due date is 15.12.`

Regex basics

Operation	Regular expression	Input string
concatenation	`foobar`	`Matches foobar but not foo or bar.`
disjunction	`this|that`	`Matches this or that.`
closure	`like.* apples`	`I like apples, Peter likes apples.`
		`Mary also likesALKFHEDL apples.`
	`like. apples`	`Mary also likesALKFHEDL apples.`
parentheses	`(He|She) likes`	`She likes apples. He likes apples.`
	`(He|She).*(very )*much\.`	`She likes apples very very much.`

- More complicated patterns can be expressed via concatenation, disjunction, repetition and scope.
- Precedence in descending order.

Quantifiers

Character	Matches
`*`	0 or more instances of preceding char
`+`	1 or more instances of preceding char
`?`	0 or one instance of preceding char
`{m}`	exactly `m` instances of preceding char
`{m,n}`	`m` through `n` instances of preceding char
`{m,}`	`m` or more instances of preceding char
`{,n}`	up to `n` instances of preceding char
`?`	add to a quantifier to match ungreedy

- Quantifiers match greedily by default (i.e. the longest string possible).
  - Ex: `^begin.*end` will match 'begin bla bla end bla end'.
    `^begin.*?end` will match 'begin bla bla end bla end'.

Groups, ranges and character classes

Character	Matches	Example RE	Matches
`.`	Any character, except `\n`	`like.`	`likes like! like  like`
`(a|b)`	a or b	`(you|me)`	`you or me`
`[ab]`	Character range	`202[01]`	`2019 2020 2021 2022`
`[a-z]`	Character range	`[A-Z][a-z]*`	Capitalized words
`[0-9]`	Digit range	`20[0-9]{2}`	Years in the 21st century
`[^ab]`	Any character but (negation)	`20[^0][01]`	`2000 2010 2020 2025 2031`

- Quantifiers, ranges and other shortcuts improve expressiveness.
  Ex: `[A-E]+` is shorthand for `(A|B|C|D|E)(A|B|C|D|E)*`.
- More character classes (sometimes) available.
  Ex: `\w` for words (`[A-Za-z0-9_]`), or `\d` for digits (`[0-9]`), `\a`, `\s`, ...

Anchors and other special characters

Character	Matches	Example RE	Matches
`^`	Beginning of a string	`^New`	New research in this field
`$`	End of a string	`[A-Za-z]+!$`	A breakthrough! Finally!
`\n`	Newline		
`\t`	Tab		
…	…		

- Strings can stretch multiple lines.
- Character encodings can sometimes cause problems. Stick to UTF-8.

Regex syntaxes

- More elaborate regex syntaxes also support positive and negative lookahead/lookbehind, conditionals and group references.
  Ex: `^(?!.*word).*$` matches lines not containing a word.
- Different syntaxes (basic, extended, perl, vim, ...) mostly similar with regard to basic features.
- Perl-compatible regular expressions (PCRE) is the de-facto standard.
- Most regex implementations feature switches to invert the search pattern, to ignore case, and more.

Example: Valid RFC-822 email addresses

http://www.ex-parrot.com/ pdw/Mail-RFC822-Address.html

Remarks

- Writing a regular expression is like writing a program.
- Requires understanding the programming model.
- Can be easier to write than read.
- Can be difficult to debug.

➥ Break up problems into smaller pieces. Try not to do everything in one large regex. Comment liberally.

➥ Regular expression tools help (e.g. regex101.com).

➥ Pin a cheatsheet to your office wall.

Remarks

- Regexes are a powerful tool.
- Easy to grasp, complex to master.

- Using them in applications can be complex and error-prone.
- Regular expressions are not parsers.

> *“Some people, when confronted with a problem, think ‘I know, I’ll use regular expressions.’ Now they have two problems.”*
>
> *Emacs newsgroup*

Next lecture: Web scraping basics

References

References

- Barrett, D. J. (2016). *Linux Pocket Guide*. Third edition. Beijing ; Boston: O'Reilly.
- Chacon, S. and B. Straub (2014). *Pro Git*. Apress.
- Fitzgerald, M. (2012). *Introducing Regular Expressions: Unraveling Regular Expressions, Step-by-Step*. 1. ed. Beijing: O'Reilly.
- Matloff, N. (2011). *The Art of R Programming: A Tour of Statistical Software Design*. No Starch Press.
- Robbins, A. (2000). *Sed & Awk Pocket Reference*. 1st ed. Beijing ; Sebastopol, CA: O'Reilly & Associates.
- Robbins, A. (2010). *Bash Pocket Reference*. 1st ed. Beijing ; Sebastopol, CA: O'Reilly.
- Shotts, W. E. (2019). *The Linux Command Line: A Complete Introduction*. Second edition. San Francisco: No Starch Press.

Python resources

Why Python?


```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
>>>
```

What (else) can Python be used for?

- Almost anything you can do in Stata, R, Gauss, Matlab or similar software. Library support is growing.
- Data management, analysis, numerics, graphs, structural modelling etc. (e.g. `scipy`, `pandas`, `numpy`, `matplotlib`, `seaborn`).
- Symbolic math (e.g. `sympy`, `Sage`).
- Geospatial work (e.g. `QGIS`).
- Text analysis and language processing (e.g. `nltk`, `spacy`, `gensim`).
- Create a website or blog (e.g. `django`, `hyde`, `sphinx`).
- Directly access many APIs (e.g. Twitter).
- Automate pretty much anything (e.g. experiments, data collection).

- In recent years, R has been extended to many of these domains.

Python resources

- Relevant modules.
  - `requests`, `bs4/BeautifulSoup`, `mechanize/mechanicalsoup`, `selenium`
  - `Scrapy` provides a complete framework for more complex projects.
  - `csv`, `re`, `pickle`, `pprint`, `pandas`, `random`, `itertools`, `pickle`, …
- Learning the language.
  - *A byte of Python* is free. *The Quick Python Book* or *Dive into Python* offer a denser treatment.
  - O'Reilly: *Learning Python*/*Programming Python*/*Pocket reference*, *Web scraping with Python*.
  - *Automate the boring stuff with Python* and other No Starch Press books for inspiration.
  - Plenty of video lectures and courses online. Stackoverflow helps.
  - (For Git: *ProGit* is free and really all you need.)
- Read about basic types, syntax, look at a few examples, then start a simple project.