

# 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

1. Economics and computation
2. Programming languages
3. Version control
4. Command line interface
5. Regular expressions

# 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, accessing APIs for data, text processing and analysis, geospatial analysis, OCR, automation, ...
  - Time complexity of algorithms, computational cost, databases, ...

- 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 corpora:  
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 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.

- 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.
- Sometimes IDEs with GUI may be more convenient.
- Features: Efficient text editing, diff, syntax checking, formatting, completion, ...



# A possible setup

```
emacs@helge-x250 ~
17 \begin{frame}[A note on operating systems]
18 \begin{itemize}
19 \item MacOS or Linux offer built-in access to a Unix shell (Bash).
20 \item Further software is managed via a package management system and
21 distributed via software repositories.
22 \item On Linux, use your package manager to install anything you require.
23 \item On MacOS, familiarize yourself with Homebrew. Install itern2 if you
24 want a function terminal.
25 \end{itemize}
26 \begin{itemize}
27 \item For Windows, many tools are not available or cumbersome to use.
28 Dependency resolution can be a nightmare.
29 \item Windows does not provide proper access to a Unix shell.
30 \item Even reliably installing Python was a chore until recently (now use
31 Anaconda).
32 \end{itemize}
33 \end{frame}
34
35 \begin{frame}[fragile][Command line interpreters and shells]
36 \begin{itemize}
37 \item An interface that lets you interact with your computer.
38 \item A CLI using a programming language that allows
39 you to execute programs and scripts.
40 \item Unix-based operating systems (Linux, MacOS) have Bash pre-installed.
41 \item Windows has cmd (or PowerShell). These are not a viable
42 replacement. Cygwin or WSL may be. Git Bash is incomplete.
43 \end{itemize}
44 \begin{itemize}
45 \item Some examples:
46 \begin{minted}[fontsize=\footnotesize]{bash}
47 cd someDir/subDir # navigate to a folder
48 cd .. # navigate to parent directory
49 cd # return to your home folder
50 ls # list directory contents
51 A # start the # console
52 \end{minted}
53 \item \textit{CTRL + C} aborts a process, \textit{CTRL + D} quits.
54 \end{itemize}
55 \end{frame}
56
57 \begin{frame}[fragile][Examples]
58 \begin{itemize}
59 \item Some examples:
60 \begin{minted}[fontsize=\footnotesize]{bash}
61 via myscript.r # edit your R script with vim
62 R -f myscript.r # execute your R script
63 python myscript.py # execute your python script
64 git add myscript.py # stage file for version control
65 git commit myscript.py # stage file for version control
66 man ssh # display manual pages for the ssh program
67 ssh myusername@13.438.14.673 # secure shell login to your remote server
68 \end{minted}
69 \end{itemize}
70 \begin{itemize}
71 \item Sounds tedious? It is. But it can also be extremely powerful.
72 \end{itemize}
73 \end{frame}
```

## Examples

- Some examples:  

```
vim myscript.r # edit your R script with vim
R -f myscript.r # execute your R script
python myscript.py # execute your python script
git add myscript.py # stage file for version control
git commit myscript.py # stage file for version control
man ssh # display manual pages for the ssh program
ssh myusername@13.438.14.673 # secure shell login to your remote server
```
- Sounds tedious? It is. But it can also be extremely powerful.
- Convert all your pdf files in a folder to text and search them.  

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

## A note on text editors

- A script is a set of *plain* text instructions, fed to an interpreter.
- R scripts usually have the suffix .r, Python .py, Shell .sh.
- Much of our work involves working with text files.
- Some text editor is required. A good text editor makes working with text much easier and faster.
- Too many options to list. All are better than Notepad.
- A few suggestions: VS Code, Sublime Text, Atom, Notepad++.
- Learning Vim or Emacs requires you to invest some time.
- Text editors allow you to integrate your work.
- Sometimes IDEs with GUI may be more convenient.
- Features: Efficient text editing, syntax checking, completion, ...

# ... that is universal

```
emacs@helge-x250 ~
10 ## Get maxpages and other information for iteration
11 response <- fromJSON(url, flatten = TRUE)
12 response$pages
13 maxpages <- response$pages$pages
14 records <- response$pages$total
15 columns <- ncol(response$loans)
16
17 ## Open csv, write header
18 header <- names(response$loans)
19 write.table(t(header), file = "data/kliva.csv", sep = ";",
20            col.names = FALSE, row.names = FALSE)
21
22 ## Or collect in data frame (don't do this for large jobs)
23 data <- data.frame(matrix(nrow = 0, ncol = columns))
24 names(data) <- header
25
26 ## Simple helper function to flatten columns
27 unnest <- function(col) paste(unlist(col), collapse = ", ")
28
29 ## Iterate over pages, limit to first three
30 for (p in seq(1, maxpages, by = 1)[1:3]) {
31   ## Info
32   print(paste0(p, "/", maxpages))
33
34   ## Append page to url
35   pquery <- paste0(url, "&page=", p)
36
37   ## Get data, assert completeness
38   loans <- fromJSON(pquery, flatten = TRUE)$loans
39   stopifnot(nrow(loans) == pagelength)
40   stopifnot(ncol(loans) == columns)
41
42   ## Fix nested list columns ... or just use data.table::fwrite()
43   ## str(loans)
44   loans$tags <- sapply(loans$tags, unnest)
45   ## loans$themes <- sapply(loans$themes, unnest) # missing for older records
46   loans$description_languages <- sapply(loans$description_languages, unnest)
47   ## str(loans)
48
49   ## Collect loans in data frame
50   data <- rbind(data, loans)
51
52   ## Append to file
53   write.table(loans, "data/kliva.csv", sep = ";", append = TRUE,
54             col.names = FALSE, row.names = FALSE)
55 }
56
57 ## head(data)
58 ## dim(data)
59
60 ## They work for you, can Example
61 apikey <- "C3WqTbTKABdQVqdBKkja"
62 base <- "https://www.theworkforyou.com/api/"
63 format <- "%s"
64
65 ## 5.0 rest-kliva.r ESS[1] [R] pLyf1m4keHwT[0 0]eHmcKJL
66 loading line: apikey <- "C3WqTbTKABdQVqdBKkja"
67
68 6 Vulnerable Groups
69 location.country_code location.country location.town location.geo_level
70 1 KH Cambodia Kampong Chan town
71 2 VN Vietnam Thanh Hoà town
72 3 VN Vietnam Thanh Hoà town
73 4 VN Vietnam Thanh Hoà town
74 5 VN Vietnam Thanh Hoà town
75 6 VN Vietnam Thanh Hoà town
76 location.geo.pairs location.geo.type
77 1 12 105.5 point
78 2 19.806692 105.785182 point
79 3 19.806692 105.785182 point
80 4 19.436971 105.374762 point
81 5 19.806692 105.785182 point
82 6 19.806692 105.785182 point
83
84 > [1] 20 25
85
86 >>>>>>> query <- paste0("country_code=", country, "&",
87 + "sectors=", sector, "&",
88 + "borrower_type=", type, "&",
89 + "status=", status, "&",
90 + "sort_by=", sortby)
91 url <- paste0(baseurl, method, query)
92 response <- fromJSON(url, flatten = TRUE)
93 response$pages
94 maxpages <- response$pages$pages
95 records <- response$pages$total
96 columns <- ncol(response$loans)
97 > $page
98 [1] 1
99
100 $total
101 [1] 3361
102
103 $page_size
104 [1] 20
105
106 $pages
107 [1] 169
108
109 >>>> header <- names(response$loans)
110 write.table(t(header), file = "data/kliva.csv", sep = ";",
111            col.names = FALSE, row.names = FALSE)
112 unnest <- function(col) paste(unlist(col), collapse = ", ")
113 for (p in seq(1, maxpages, by = 1)[1:3]) {
114   print(paste0(p, "/", maxpages))
115   pquery <- paste0(url, "&page=", p)
116   loans <- fromJSON(pquery, flatten = TRUE)$loans
117   stopifnot(nrow(loans) == pagelength)
118   stopifnot(ncol(loans) == columns)
119   loans$tags <- sapply(loans$tags, unnest)
120   loans$description_languages <- sapply(loans$description_languages, unnest)
121   write.table(loans, "data/kliva.csv", sep = ";", append = TRUE,
122             col.names = FALSE, row.names = FALSE)
123 }
124
125 apikey <- "C3WqTbTKABdQVqdBKkja"
126 [1] "1/169"
127 [1] "2/169"
128 [1] "3/169"
129 >>>

```

# ... that is universal

<p>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.<sup>\footnote{Using PISA data from secondary schools in Switzerland, \cite{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.</sup></p>	<p>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.<sup>\footnote{Using PISA data from secondary schools in Switzerland, \cite{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.</sup></p>
<p>174 % NEU: Bei bitte lesen 175 % changed some small things. OK for me. 176 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 <b>classrooms</b> is ignorable with respect to SN status \cite{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 \cite{carrell2010}. We re-sample classes and thereby assign SN students randomly to <b>classrooms</b>, 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.</p>	<p>172 % NEU: Bei bitte lesen 173 % 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 <b>classroom</b> is ignorable with respect to SN status \cite{chettyEtal2011}. We also conduct this test to check for ignorable assignment of SN students to <b>specific</b> 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 \cite{carrell2010}. We re-sample classes, randomly assign SN students to <b>classes</b>, 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.</p>
<p>178 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.<sup>\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 (SOL, \textit{Statistik der Lernenden} in German) for the years 2012-2015. Importantly, the SOL 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 no</sup></p>	<p>175 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.<sup>\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 (SOL, \textit{Statistik der Lernenden} in German) for the years 2012-2015. Importantly, the SOL 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 no</sup></p>
<p>A: ● 104k xSource/manuscript_R1_v3.tex 77:0 37% LF UTF-8 LaTeX/FPS 1</p>	<p>B: ● 104k x010/manuscript_R1_v2.tex 77:0 37% LF UTF-8 LaTeX/FPS 11</p>
<p>type ? for help</p>	
<p>*Edit Control Panel* diff. 4 of 15 Quick Help</p>	
<p>Auto-refining is ON</p>	

RStudio

Project: (None)

Go to file/function

Source on Save

Run

Source

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

```

Console

```

> 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)
>

```

Files Plots Packages Help Viewer

10:37 (Top Level)

R Script

Environment History Presentation

Import Dataset Clear

Global Environment

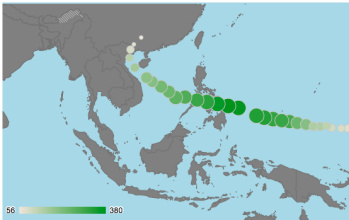
Data

dat 33 obs. of 11 variables

Values

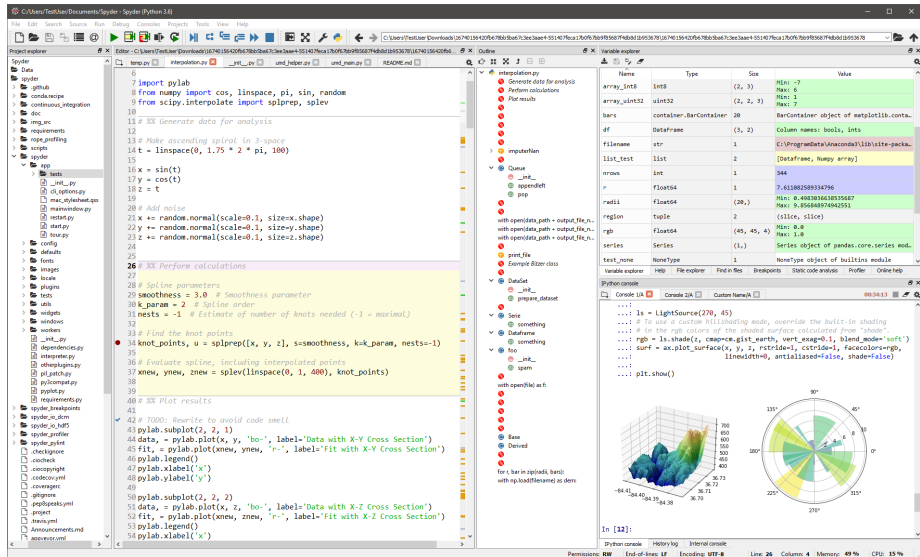
M List of 3

url "http://www.gdacs.org/Cyclones/report.aspx?eventId=41058&..."



56 380

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



# Version control

---

## "FINAL".doc



FINAL.doc!



FINAL\_rev.2.doc



FINAL\_rev.6.COMMENTS.doc



FINAL\_rev.8.comments5.  
CORRECTIONS.doc



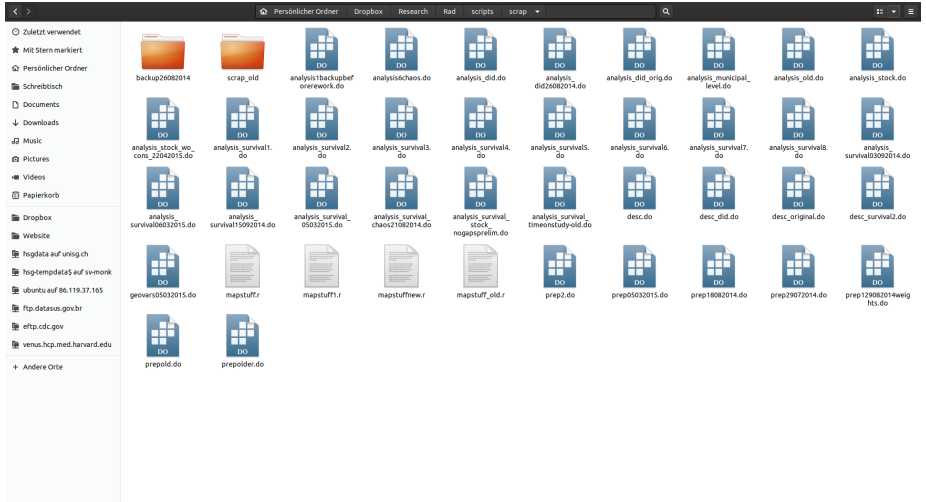
FINAL\_rev.18.comments7.  
corrections9.MORE.30.doc



FINAL\_rev.22.comments49.  
corrections.10.##\$%WHYDID  
ICOMETOGRADSCHOOL?????.doc

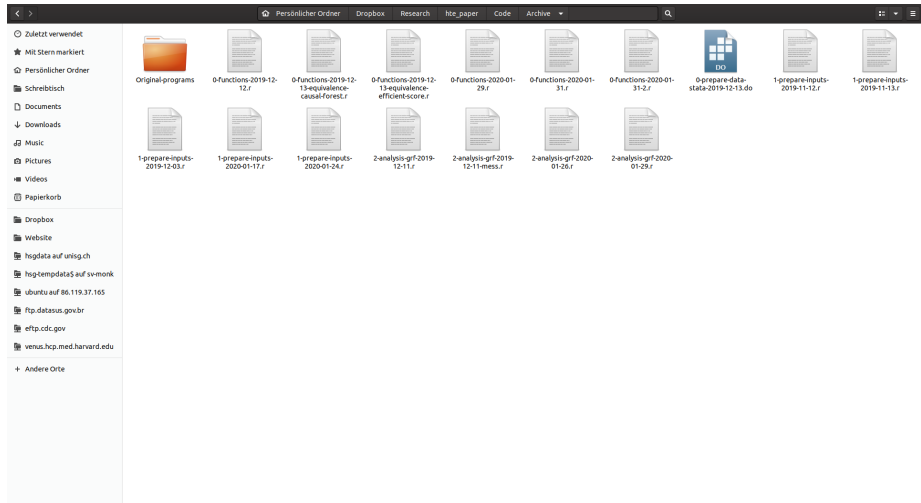
JORGE CHAM © 2012

# Familiar?





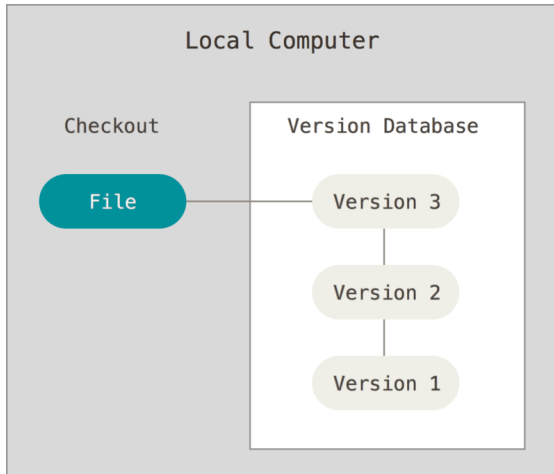
# Familiar?



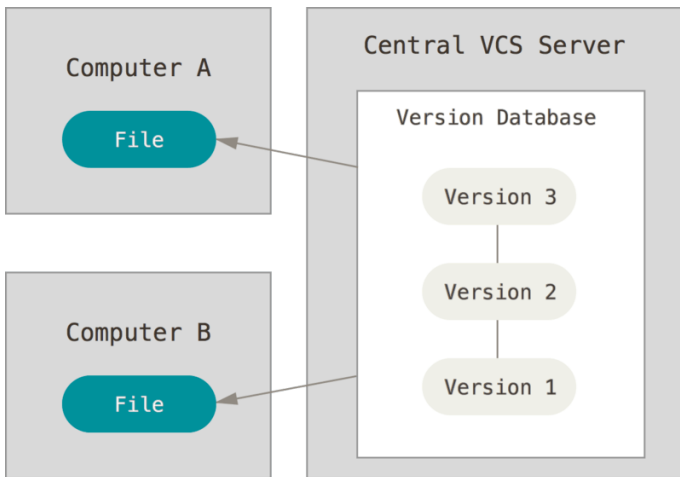
# Tracking different versions

- 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.

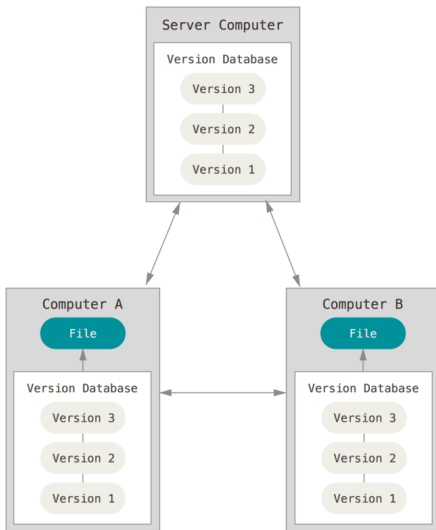
# Local version control



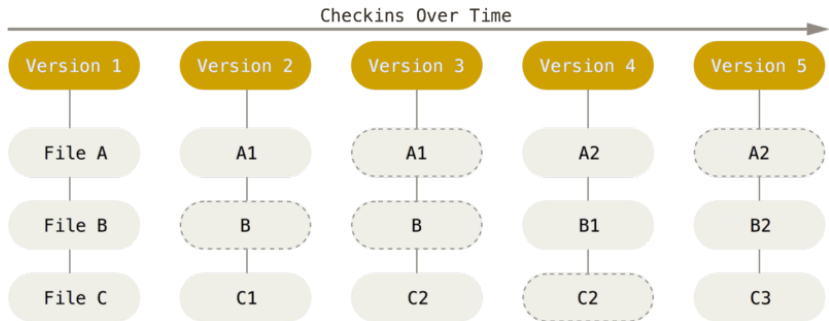
# Centralized version control



# Distributed version control

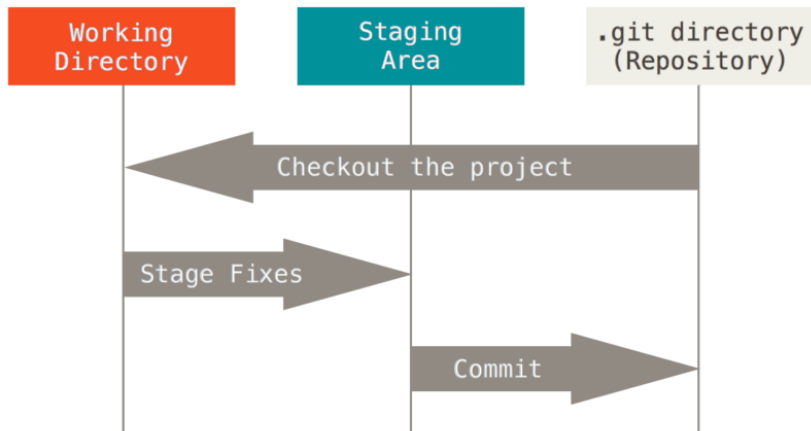


# Distributed version control - snapshots



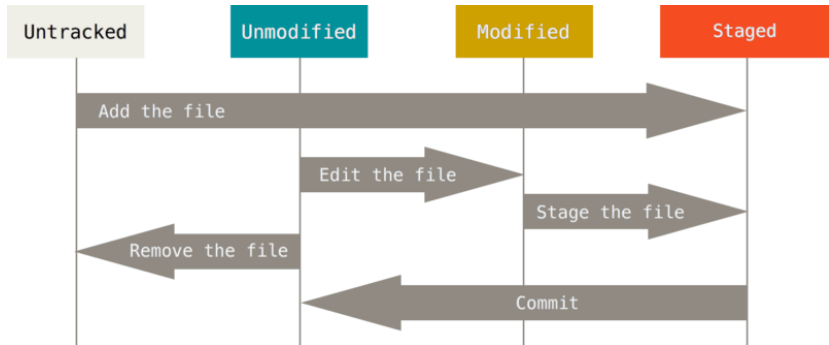
- Git is the dominant *distributed* version control system today.
- Developed by Linus Torvalds, the creator of the Linux kernel.
- Can be used on the command line. GUIs/editor plugins available. Integrated into RStudio.
- Fast, good support for non-linear development (thousands of parallel branches).
- 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 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





# 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.

[illegible]

# 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 controversies! 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 controversies! 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 punctuation, 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
<code>input</code>	This <code>input</code> string is short.
<code>15</code>	The due date is <code>15.12</code> .

# Matching string literals

Regular expression	Input string
<code>input</code>	This <code>input</code> string is short.
<code>15</code>	The due date is <code>15.12.</code>
<code>15.12</code>	The due date is <code>15.12.</code>

# Matching string literals

Regular expression	Input string
<code>input</code>	This <code>input</code> string is short.
<code>15</code>	The due date is <code>15.12</code> .
<code>15.12</code>	The due date is <code>15.12</code> .
but:	
<code>15.12</code>	The due date is <code>15712</code> .
match <code>.</code> literal:	
<code>15\.12</code>	The due date is <code>15712</code> .
<code>15\.12</code>	The due date is <code>15.12</code> .



# Regex basics

Operation	Regular expression	Input string
concatenation	<code>foobar</code>	Matches <code>foobar</code> but not <code>foo</code> or <code>bar</code> .
disjunction	<code>this that</code>	Matches <code>this</code> or <code>that</code> .
closure	<code>like.* apples</code>	I like apples, Peter likes apples. Mary also likesALKFHEDL apples.
	<code>like. apples</code>	Mary also likesALKFHEDL apples.
parentheses	<code>(He She) likes</code>	She likes apples. He likes apples.
	<code>(He She).*(very )*much\.</code>	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
<code>^</code>	Beginning of a string	<code>^New</code>	New research in this field
<code>\$</code>	End of a string	<code>[A-Za-z]+!\$</code>	A breakthrough! Finally!
<code>\n</code>	Newline		
<code>\t</code>	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.

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

[illegible]

# 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](https://regex101.com)).
  - ➡ Pin a cheatsheet to your office wall.

- 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

---

```
>>> 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.