Цифровая экономика

Искусственный интеллект в оценочной деятельности

Практическое руководство по разработке систем поддержки принятия решений оценщиками с использованием языков программирования R и Python

К. А. Мурашев

4 сентября 2021 г.

УДК 519(2+8+682)+004.891.2+330.4+338.5
ББК 16.6+22(16+17)+65.25
ГРНТИ 27.43.51+28.23.35+28.23.29+28.23.37+83.03.51
М91
Искусственный интеллект в оценочной деятельности: практическое руководство по разработке систем поддержки принятия решений оценщиками с использованием языков программирования R и Python / К. А. Мурашев — Inkeri, Санкт-Петербург, 12 августа 2021 г. – 4 сентября 2021 г., 67 с.

В процессе разработки данного материала равно как и программного кода автор использовал операционную систему Kubuntu [9]. Для подготовки данного материала использовался язык TEX [66] с набором макрорасширений LATEX 2ε [67]. Конкретная техническая реализация заключается в использовании дистрибутива TexLive [68], редактора LyX [44], компилятора PdfLaTeX и системы цитирования BibLaTeX/Biber. Исходный код и дополнительные файлы, необходимые для его компиляции, доступны по ссылке на портале Github [53], а также по запасной ссылке [54].

Материал подготовлен в форме гипертекста: ссылки на ресурсы, размещённые в информационно-телекоммуникационной сети «Интернет» [122], выделены синим (blue) цветом, внутренние перекрёстные ссылки выделены красным (red) цветом, библиографические ссылки выделены зелёным (green) цветом. При подготовке данного материала использовался шаблон KOMA-Script Book [38]. В целях облегчения понимания согласования слов в сложноподчинённых предложениях либо их последовательности в тексте реализована графическая разметка, позволяющая понять

структуру предложения: слова, согласованные между собой внутри предложения, подчёркнуты одинаковыми линиями, данное решение применяется только в тех предложениях, в которых, по мнению автора, возможно неоднозначное толкование в части согласования слов внутри него.

Данный материал выпускается в соответствии с философией *Rolling Release [84]*, что означает что он будет непрерывно дорабатываться по мере обнаружения ошибок и неточностей, а также в целях улучшения внешнего вида. Идентификатором, предназначенным для определения версии материала, служат её номер и дата релиза, указанные на титульном листе, а также в колонтитулах. История версий приводится в таблице 0.1 на следующей странице–4. Актуальная версия перевода в формате PDF доступна по ссылке [53], а также по запасной ссылке [54].

В целях соответствия принципам устойчивого развития [34, 89], установленным в частности Стратегией The European Green Deal [57] и являющимся приоритетными для Единой Европы [28, 12, 77], а также содействия достижению углеродной нейтральности [71] рекомендуется использовать материал исключительно в электронной форме без распечатывания на бумаге.

Для связи с автором данного перевода можно использовать

- любой клиент, совместимый с протоколом Tox [59, 90], Tox ID = 2E71 CA29 AF96 DEF6 ABC0 55BA 4314 BCB4 072A 60EC C2B1 0299 04F8 5B26 6673 C31D 8C90 7E19 3B35;
- адрес электронной почты: kirill.murashev@tutanota.de;
- https://www.facebook.com/murashev.kirill/ [1];

Реквизиты для оказания помощи проекту.

Тинькоф: +79219597644

BTC: bc1qjzwtk3hc7ft9cf2a3u77cxfklgnw93jktyjfsl?time=1627474534&exp=86400

ETH:

Monero: 45ho 6Na3 dzoW DwYp 4ebD BXBr 6CuC F9L5 NGCD ccpa w2W4 W15a fiMM dGmf dhnp e6hP JSXk 9Mwm o9Up kh3a ek96 LFEa BZYX zGQ

USDT: 0x885e0b0E0bDCFE48750Be534f284EFfbEf6d247C

EURT: 0x885e0b0E0bDCFE48750Be534f284EFfbEf6d247C

CNHT: 0x885e0b0E0bDCFE48750Be534f284EFfbEf6d247C

История версий

Таблица 0.0.1: История версий материала

№	Номер версии	Дата	Автор	Описание
0	1	2	3	4
1	0.0001.0001	2021-08-14	KAM	Initial

Оглавление

List of Algorithms

Список иллюстраций

Список таблиц

Список литературы

[1] URL: https://www.facebook.com/murashev.kirill/ (дата обр. 28.07.2021).

[2] Royal Institution Surveyors of Chartered (RICS). *RICS Valuation — Global Standards*. English. UK, London: RICS, 28 нояб. 2019. URL: https://www.rics.org/eu/upholding-professional-standards/sector-standards/valuation/red-book/red-book-global/ (дата обр. 10.06.2020).

[3] *Apache 2.0*. URL: http://licenseit.ru/wiki/index.php/Apache_License_version_2.0#.D0.A2.D0.B5.D0.BA.D1.81.D1.82_.D0.BB.D0.B8.D1.86.D0.B5.D0.BD.D0.B7.D0.B8.D0.B8 (дата обр. 17.08.2021).

[4] Scott Chacon. *Pro Git book*. Перевод на русский язык. URL: https://git-scm.com/book/ru/v2 (дата обр. 25.08.2021).

[5] Creative Commons. *Creative Commons Attribution-ShareAlike 4.0 International*. нояб. 2013. URL: https://creativecommons.org/licenses/by-sa/4.0/legalcode.

[6] Creative Commons. *Creative Commons Attribution-ShareAlike 4.0 International RUS*. нояб. 2013. URL: https://creativecommons.org/licenses/by-sa/4.0/legalcode.ru.

[7] Microsoft Corporation. *Microsoft Excel*. Английский. URL: https://www.microsoft.com/en-us/microsoft-365/excel (дата обр. 20.08.2021).

[8] CorVVin. *Хеш-функция, что это такое?* URL: https://habr.com/en/post/534596/ (дата обр. 25.08.2021).

[9] Kubuntu devs. *Kubuntu official site*. Kubuntu devs. URL: https://kubuntu.org/ (дата обр. 17.08.2021).

[10] KDE e.V. *Plasma. KDE community*. Английский. KDE e.V. URL: https://kde.org/plasma-desktop/ (дата обр. 19.08.2021).

[11] *Ed25519*. URL: https://ed25519.cr.yp.to/ (дата обр. 04.09.2021).

[12] Institute Greater for a Europe. *Institute for a Greater Europe official site*. URL: https://www.institutegreatereurope.com/ (дата обр. 15.04.2021).

[13] StatSoft Europe. *Statistica: official site*. URL: https://www.statistica.com/en/ (дата обр. 24.08.2021).

[14] Apache Software Foundation. *Apache LicenseVersion 2.0*. Английский. URL: https://www.apache.org/licenses/LICENSE-2.0 (дата обр. 17.08.2021).

[15] Python Software Foundation. Английский. Python Software Foundation. URL: https://www.python.org/ (дата обр. 17.08.2021).

[16] The Apache Software Foundation. *OpenOffice Calc*. URL: https://www.openoffice.org/product/calc.html (дата обр. 20.08.2021).

[17] The Document Foundation. *LibreOffice Calc*. Английский. URL: https://www.libreoffice.org/discover/calc/ (дата обр. 20.08.2021).

[18] The IFRS Foundation. *IFRS 13 Fair Value Measurement*. UK, London: The IFRS Foundation, 31 янв. 2016. URL: http://eifrs.ifrs.org/eifrs/bnstandards/en/IFRS13.pdf (дата обр. 10.06.2020).

[19] Geeksforgeeks. *Difference between RSA algorithm and DSA*. URL: https://www.geeksforgeeks.org/difference-between-rsa-algorithm-and-dsa/ (дата обр. 04.09.2021).

[20] *GeoGebra official site*. URL: https://www.geogebra.org/ (дата обр. 26.08.2021).

[21] *Git Download for Windows*. URL: https://git-scm.com/download/win (дата обр. 29.08.2021).

[22] *Git install on macOS*. URL: https://git-scm.com/download/mac (дата обр. 29.08.2021).

[23] *Git official site*. URL: https://git-scm.com/ (дата обр. 19.08.2021).

[24] *Git на сервере — Протоколы*. URL: https://git-scm.com/book/ru/v2/Git-%D0%BD%D0%B0-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%B5-%D0%9F%D1%80%D0%BE%D1%82%D0%BE%D0%BA%D0%BE%D0%BB%D1%8B (дата обр. 03.09.2021).

[25] *GitHub Desktop*. URL: https://desktop.github.com/ (дата обр. 19.08.2021).

[26] *Github site*. URL: https://github.com/ (дата обр. 03.09.2021).

[27] Google. *Google Sheets*. URL: https://www.google.com/sheets/about/ (дата обр. 20.08.2021).

[28] Lisbon-Vladivostok Work group. *Initiative Lisbon-Vladivostok*. URL: https://lisbon-vladivostok.pro/ (дата обр. 15.04.2021).

[29] *Homebrew*. URL: https://brew.sh/ (дата обр. 29.08.2021).

[30] IBM. *SPSS: official page*. URL: https://www.ibm.com/products/spss-statistics (дата обр. 24.08.2021).

[31] IHS Global Inc. *Eviews: official site*. URL: https://www.eviews.com/home.html (дата обр. 24.08.2021).

[32] SAS Institute Inc. *SAS: official site*. URL: https://www.sas.com/en_us/home.html (дата обр. 24.08.2021).

[33] Intel. *Процессор Intel® Core™ i7-7500U*. Русский. тех. отч. URL: https://ark.intel.com/content/www/ru/ru/ark/products/95451/intel-core-i7-7500u-processor-4m-cache-up-to-3-50-ghz.html (дата обр. 19.08.2021).

[34] Investopedia. *Sustainability*. URL: https://www.investopedia.com/terms/s/sustainability.asp (дата обр. 15.04.2021).

[35] ISO. *Office Open XML*. URL: https://standards.iso.org/ittf/PubliclyAvailableStandar
c071692_ISO_IEC_29500-4_2016.zip (дата обр. 20.08.2021).

[36] ISO/IEC. *ISO/IEC 10746-2:2009. Information technology "— Open distributed processing "— Reference model: Foundations — Part 2*. English. под ред. ISO/IEC. Standard. ISO/IEC, 15 дек. 2009. URL: http://docs.cntd.ru/document/431871894 (дата обр. 01.03.2021).

[37] ISO/IEC. *ISO/IEC 2382:2015. Information technology — Vocabulary*. English. под ред. ISO/IEC. ISO/EIC, 2015. URL: https://www.iso.org/obp/ui/#iso:std:iso-iec:2382:ed-1:v1:en (дата обр. 01.03.2021).

[38] Markus Kohm. *koma-script – A bundle of versatile classes and packages*. 1994–2020. URL: https://ctan.org/pkg/koma-script (дата обр. 28.01.2021).

[39] *LaTeXDraw official page*. URL: http://latexdraw.sourceforge.net/ (дата обр. 26.08.2021).

[40] Licenseit.ru. *GNU General Public License*. URL: http://licenseit.ru/wiki/index.php/GNU_General_Public_License (дата обр. 23.08.2021).

[41] Licenseit.ru. *GNU General Public License version 2*. URL: http://licenseit.ru/wiki/index.php/GNU_General_Public_License_version_2 (дата обр. 23.08.2021).

[42] Licenseit.ru. *Python License version 2.1*. URL: http://licenseit.ru/wiki/index.php/Python_License_version_2.1 (дата обр. 23.08.2021).

[43] StataCorp LLC. *Stata: official site*. URL: https://www.stata.com/ (дата обр. 24.08.2021).

[44] *LyX official site*. URL: https://www.lyx.org/ (дата обр. 28.01.2021).

[45] Machinelearning.ru. *Нормальное распределение*. URL: http://www.machinelearning.ru/wiki/index.php?title=%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%80%D0%B0%D1%81%D0%BF%D1%80%D0%B5%D0%B4%D0%B5%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5 (дата обр. 02.03.2021).

[46] Machinelearning.ru. *Параметрические статистические тесты*. URL: http://www.machinelearning.ru/wiki/index.php?title=%D0%9A%D0%B0%D1%82%D0%B5%D0%B3%D0%BE%D1%80%D0%B8%D1%8F:%D0%9F%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B5_%D1%82%D0%B5%D1%81%D1%82%D1%8B (дата обр. 02.03.2021).

[47] LLC Minitab. *Minitab: official site*. URL: https://www.minitab.com/en-us/ (дата обр. 24.08.2021).

[48] Kirill A. Murashev. R. URL: https://github.com/Kirill-Murashev/AI_for_valuers_R_source.

[49] Kirill A. Murashev. R. URL: https://web.tresorit.com/l/1Zgvt#kBA5FiYOQtverp8Rjz6gyg.

[50] Kirill A. Murashev. R. URL: https://github.com/Kirill-Murashev/AI_for_valuers_Python_source.

[51] Kirill A. Murashev. R. URL: https://web.tresorit.com/l/VGZE5#XqySAkmjYODAIcOp1ZWPmg.

[52] Kirill A. Murashev. *RICS Valuation — Global Standrards 2020. Russian translation.* TeX. 28 июля 2021. URL: https://web.tresorit.com/l/oFpJF#xr3UGoxLvszsn4vAaHtjqw.

[53] Kirill A. Murashev. *Искусственный интеллект в оценочной деятельности: практическое руководство по разработке систем поддержки принятия решений оценщиками с использованием языков программирования R и Python*. Inkeri. URL: https://github.com/Kirill-Murashev/AI_for_valuers_book.

[54] Kirill A. Murashev. *Искусственный интеллект в оценочной деятельности: практическое руководство по разработке систем поддержки принятия решений оценщиками с использованием языков программирования R и Python*. Inkeri. URL: https://web.tresorit.com/l/3xiTP#1p8pFnG_9No9izLFd09xaA.

[55] *Notepad++ site*. URL: https://notepad-plus-plus.org/ (дата обр. 29.08.2021).

[56] Linux Kernel Organization. *The Linux Kernel Archives*. Linux Kernel Organization. URL: https://www.kernel.org/ (дата обр. 26.08.2021).

[57] European Parliament. *The European Green Deal*. 15 янв. 2020. URL: https://www.europarl.europa.eu/doceo/document/TA-9-2020-0005_EN.html (дата обр. 15.04.2021).

[58] Risan Bagja Pradana. *Upgrade Your SSH Key to Ed25519*. URL: https://medium.com/risan/upgrade-your-ssh-key-to-ed25519-c6e8d60d3c54 (дата обр. 04.09.2021).

[59] Tox Project. *Tox project official site*. URL: https://tox.chat/ (дата обр. 09.03.2021).

[60] *Qt*. Английский. URL: https://www.qt.io/ (дата обр. 19.08.2021).

[61] R Foundation. *The Comprehensive R Archive Network*. URL: https://cran.r-project.org/ (дата обр. 24.08.2021).

[62] *SHA3-512 online hash function*. URL: https://emn178.github.io/online-tools/sha3_512.html (дата обр. 25.08.2021).

[63] Stackexchange. *RSA vs. DSA for SSH authentication keys*. URL: https://security.stackexchange.com/questions/5096/rsa-vs-dsa-for-ssh-authentication-keys (дата обр. 04.09.2021).

[64] Statsoft. *Solving trees*. URL: http://statsoft.ru/home/textbook/modules/stclatre.html (дата обр. 20.08.2021).

[65] PBC Studio. *RStudio official site*. Английский. URL: https://www.rstudio.com/ (дата обр. 19.08.2021).

[66] CTAN team. *TeX official site*. English. CTAN Team. URL: https://www.ctan.org/ (дата обр. 15.11.2020).

[67] LaTeX team. *LaTeX official site*. English. URL: https://www.latex-project.org/ (дата обр. 15.11.2020).

[68] *TeXLive official site*. URL: https://www.tug.org/texlive/ (дата обр. 15.11.2020).

[69] The R Foundation. *The R Project for Statistical Computing*. Английский. The R Foundation. URL: https://www.r-project.org/ (дата обр. 17.08.2021).

[70] Wikipedia. *Bash (Unix shell)*. URL: https://en.wikipedia.org/wiki/Bash_(Unix_shell) (дата обр. 02.09.2021).

[71] Wikipedia. *Carbon neutrality*. URL: https://en.wikipedia.org/wiki/Carbon_neutrality (дата обр. 15.04.2021).

[72] Wikipedia. *COVID-19 pandemic*. Английский. URL: https://en.wikipedia.org/wiki/COVID-19_pandemic (дата обр. 18.08.2021).

[73] Wikipedia. *DSA*. URL: https://ru.wikipedia.org/wiki/DSA (дата обр. 04.09.2021).

[74] Wikipedia. *ECDSA*. URL: https://ru.wikipedia.org/wiki/ECDSA (дата обр. 04.09.2021).

[75] Wikipedia. *Efficient-market hypothesis*. URL: https://en.wikipedia.org/wiki/Efficient-market_hypothesis (дата обр. 29.10.2020).

[76] Wikipedia. *Euclidean distance*. URL: https://en.wikipedia.org/wiki/Euclidean_distance (дата обр. 18.08.2021).

[77] Wikipedia. *Greater Europe*. URL: https://en.wikipedia.org/wiki/Greater_Europe (дата обр. 15.04.2021).

[78] Wikipedia. *HTTPS*. URL: https://en.wikipedia.org/wiki/HTTPS (дата обр. 03.09.2021).

[79] Wikipedia. *Kelly Johnson (engineer)*. URL: https://en.wikipedia.org/wiki/Kelly%5C_Johnson_(engineer) (дата обр. 06.11.2020).

[80] Wikipedia. *KISS principle*. URL: https://en.wikipedia.org/wiki/KISS_principle (дата обр. 06.11.2020).

[81] Wikipedia. $List_{o}f_{L}inux_{d}istributions : Debian - based$. URL: https://en.wikipedia.org/wiki/Category:Debian-based_distributions (дата обр. 26.08.2021).

[82] Wikipedia. *Office Open XML*. URL: https://ru.wikipedia.org/wiki/Office_Open_XML (дата обр. 20.08.2021).

[83] Wikipedia. *Robert$_{\text{G}}$entleman*. URL: https://en.wikipedia.org/wiki/Robert_Gentleman_(statistician) (дата обр. 25.08.2021).

[84] Wikipedia. *Rolling Release*. URL: https://ru.wikipedia.org/wiki/Rolling_release (дата обр. 28.01.2021).

[85] Wikipedia. *Ross$_{\text{I}}$haka*. URL: https://en.wikipedia.org/wiki/Ross_Ihaka (дата обр. 25.08.2021).

[86] Wikipedia. *RSA*. URL: https://ru.wikipedia.org/wiki/RSA (дата обр. 04.09.2021).

[87] Wikipedia. *SHA-3*. URL: https://ru.wikipedia.org/wiki/SHA-3 (дата обр. 26.08.2021).

[88] Wikipedia. *SSH*. URL: https://ru.wikipedia.org/wiki/SSH (дата обр. 03.09.2021).

[89] Wikipedia. *Sustainability*. English. URL: https://en.wikipedia.org/wiki/Sustainability (дата обр. 15.04.2021).

[90] Wikipedia. *Wikipedia: Tox protocol*. URL: https://en.wikipedia.org/wiki/Tox_(protocol) (дата обр. 09.03.2021).

[91] Wikipedia. *Архитектура компьютера*. Russian. URL: https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%85%D0%B8%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%BA%D0%BE%D0%BC%D0%BF%D1%8C%D1%8E%D1%82%D0%B5%D1%80%D0%B0 (дата обр. 06.08.2021).

[92] Wikipedia. *Высокоуровневый язык программирования*. URL: https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F (дата обр. 23.08.2021).

[93] Wikipedia. *Детерминированный алгоритм*. URL: https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%82%D0%B5%D1%80%D0%BC%D0%B8%D0%BD%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC (дата обр. 25.08.2021).

[94] Wikipedia. *Дискретное логарифмирование*. URL: https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BB%D0%BE%D0%B3%D0%B0%D1%80%D0%B8%D1%84%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5 (дата обр. 04.09.2021).

[95] Wikipedia. *Интегрированная среда разработки*. URL: https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8 (дата обр. 29.08.2021).

[96] Wikipedia. *Коллизия хеш-функции*. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BB%D0%BB%D0%B8%D0%B7%D0%B8%D1%8F_%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8 (дата обр. 25.08.2021).

[97] Wikipedia. *Конечное поле (поле Галуа)*. URL: https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BD%D0%B5%D1%87%D0%BD%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5 (дата обр. 04.09.2021).

[98] Wikipedia. *Непараметрическая статистика*. URL: https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BF%D0%B0%D1%80%D0%B0%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B0 (дата обр. 20.08.2021).

[99] Wikipedia. *Переменная (математика)*. URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%B2%D0%B5%D0%BB%D0%B8%D1%87%D0%B8%D0%BD%D0%B0 (дата обр. 20.08.2021).

[100] Wikipedia. *Переменная (программирование)*. URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%BC%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) (дата обр. 20.08.2021).

[101] Wikipedia. *Полнота по Тьюрингу*. URL: https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%BB%D0%BD%D0%BE%D1%82%D0%B0_%D0%BF%D0%BE_%D0%A2%D1%8C%D1%8E%D1%80%D0%B8%D0%BD%D0%B3%D1%83 (дата обр. 23.08.2021).

[102] Wikipedia. *Принцип Дирихле*. URL: https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%B8%D0%BD%D1%86%D0%B8%D0%BF_%D0%94%D0%B8%D1%80%D0%B8%D1%85%D0%BB%D0%B5_(%D0%BA%D0%BE%D0%BC%D0%B1%D0%B8%D0%BD%D0%B0%D1%82%D0%BE%D1%80%D0%B8%D0%BA%D0%B0) (дата обр. 25.08.2021).

[103] Wikipedia. *Расстояние городских кварталов*. URL: https://en.wikipedia.org/wiki/Taxicab_geometry (дата обр. 18.08.2021).

[104] Wikipedia. *Сверхвысокоуровневый язык программирования*. URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%B5%D1%80%D1%85%D0%B2%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F (дата обр. 23.08.2021).

[105] Wikipedia. *Свободная лицензия*. URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%B0%D1%8F_%D0%BB%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F (дата обр. 23.08.2021).

[106] Wikipedia. *Свободное программное обеспечение*. Русский. URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D0%BE%D0%B1%D0%BE%D0%B4%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5 (дата обр. 18.08.2021).

[107] Wikipedia. *Сильная форма Гипотезы эффективного рынка*. URL: https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%BF%D0%BE%D1%82%D0%B5%D0%B7%D0%B0_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D0%B3%D0%BE_%D1%80%D1%8B%D0%BD%D0%BA%D0%B0#%D0%A2%D1%80%D0%B8_%D1%84%D0%BE%D1%80%D0%BC%D1%8B_%D1%80%D1%8B%D0%BD%D0%BE%D1%87%D0%BD%D0%BE%D0%B9_%D1%8D%D1%84%D1%84%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%BE%D1%81%D1%82%D0%B8 (дата обр. 18.08.2021).

[108] Wikipedia. *Сценарный язык*. URL: https://ru.wikipedia.org/wiki/%D0%A1%D1%86%D0%B5%D0%BD%D0%B0%D1%80%D0%BD%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA (дата обр. 23.08.2021).

[109] Wikipedia. *Хеш-функция*. URL: https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F (дата обр. 25.08.2021).

[110] Wikpedia. *Эллиптическая кривая*. URL: https://ru.wikipedia.org/wiki/%D0%AD%D0%BB%D0%BB%D0%B8%D0%BF%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B0%D1%8F_%D0%BA%D1%80%D0%B8%D0%B2%D0%B0%D1%8F (дата обр. 04.09.2021).

[111] *Xcode page*. URL: https://developer.apple.com/xcode/ (дата обр. 29.08.2021).

[112] *Как запустить Bash скрипт в Linux*. URL: https://wiki.merionet.ru/servernye-resheniya/63/kak-zapustit-bash-skript-v-linux/ (дата обр. 02.09.2021).

[113] Кирилл Кринкин. *Введение в архитектуру ЭВМ и элементы ОС. Курс лекций*. Русский. Computer Science Center. URL: https://www.youtube.com/watch?v=FzN8zzMRTlw&list=PLlb7e2G7aSpRZ9wDzXI-VYpk59acLFOIr (дата обр. 23.08.2021).

[114] Артём Матяшов. *Git. Большой практический выпуск*. Русский. URL: https://www.youtube.com/watch?v=SEvR78OhGtw (дата обр. 03.09.2021).

[115] связи и массовых коммуникаций Российской Федерации Министерство цифрового развития. *Свободное программное обеспечение в госорганах*. Русский. URL: https://www.gnu.org/philosophy/free-sw.ru.html (дата обр. 18.08.2021).

[116] Фонд свободного программного обеспечения. *Что такое свободная программа?* Русский. Фонд свободного программного обеспечения. URL: https://www.gnu.org/philosophy/free-sw.ru.html (дата обр. 18.08.2021).

[117] Программирование на C и C++. Онлайн справочник программиста на C и C++. *Оператор*. URL: http://www.c-cpp.ru/books/operatory (дата обр. 20.08.2021).

[118] Виталий Радченко. *Открытый курс машинного обучения. Тема 5. Композиции: бэггинг, случайный лес*. URL: https://habr.com/en/company/ods/blog/324402/ (дата обр. 20.08.2021).

[119] Министерство финансов России. *Международный стандарт финансовой отчётности (IFRS) 13 «Оценка справедливой стоимости»*. с изменениями на 11 июля 2016 г. Russian. Russia, Moscow: Минфин России, 28 дек. 2015. URL: https://normativ.kontur.ru/document?moduleId=1&documentId=326168#l0 (дата обр. 10.06.2020).

[120] Министерство цифрового развития Российской Федерации. *Национальная программа «Цифровая экономика Российской Федерации»*. 29 окт. 2020. URL: https://digital.gov.ru/ru/activity/directions/858/ (дата обр. 29.10.2020).

[121] Министерство экономического развития РФ. *Федеральные стандарты оценки*. URL: https://www.consultant.ru/document/cons_doc_LAW_126896/.

[122] Российская Федерация. *Федеральный Закон «Об информации, информационных технологиях и о защите информации»*. 149-ФЗ. Russian. Russia, Moscow, 14 июля 2006. URL: https://normativ.kontur.ru/document?moduleId=1&documentId=376603&cwi=22898 (дата обр. 07.07.2020).

[123] Российская Федерация. *Федеральый закон «Об оценочной деятельности в Российской Федерации»*. 29 июля 1998. URL: https://normativ.kontur.ru/document?moduleId=1&documentId=396506&cwi=7508 (дата обр. 18.08.2021).

Глава 1.

Предисловие

«Лучший способ в чём-то разобраться до конца — это попробовать научить этому компьютер».
Дональд Э. Кнут

Целью данной работы является попытка объединения наработок в областях оценочной деятельности и искусственного интеллекта. Автор предпринимает попытку доказать возможность применения современных технологий искусственного интеллекта в сфере оценки имущества, его эффективность и наличие ряда преимуществ относительно иных методов определения стоимости и анализа данных открытых рынков. В условиях заданного руководством России курса на цифровизацию экономики и, в особенности, на развитие технологий искусственного интеллекта [120] внедрение методов машинного обучения в повседневную практику оценщиков представляется логичным и необходимым.

Данная работа писалась в условиях распространения новой коронавирусной инфекции [72], внёсшей дополнительный вклад в процессы цифровизации во всём мире. Можно по-разному относиться к проблематике данного явления, однако нельзя отрицать его влияние на общество и технологический уклад ближайшего будущего. Повсеместный переход на технологии искусственного интеллекта, замена человеческого труда машинным, беспрецедентный рост капитализации компаний, сделавших ставку на развитие интеллектуальной собственности, делают невозможным игнорирование необходимости цифровой трансформации оценочной деятельности в России.

Актуальность предложенного автором исследования заключается во-первых в том, что оно даёт практический инструментарий, позволяющий делать обоснованные, поддающие верификации выводы на основе использования исключительно объективных информации и данных,[1] непосредственно наблюдаемых на открытых рын-

[1]По мнению автора, отличие между информацией и данными заключается в том, что под ин-

ках, без использования каких-либо иных их источников, подверженных субъективному влиянию со стороны их авторов. Во-вторых, предложенные и рассмотренные в данной работе методы обладают весьма широким функционалом, позволяющим использовать их при решении широкого круга задач, выходящих за рамки работы над конкретной оценкой. Важность обеих причин автор видит в том, что на 2021 год в России в сфере оценочной деятельности сложилась ситуация, которую можно охарактеризовать тремя состояниями:

- состояние неопределённости будущего отрасли;
- состояние интеллектуального тупика;
- состояние технологической отсталости.

Первая проблема заключается в неопределённости как правового регулирования отрасли, так и её экономики. Введённая около четырёх лет назад система квалификационных аттестатов оценщиков, на которую регулятор, заказчики и, возможно, часть самих оценщиков возлагали надежду как на фильтр, позволяющий оставить в отрасли только квалифицированных специалистов, сократить предложение оценочных услуг и, следовательно, способствовать росту вознаграждений за проведение оценки, не оправдала ожиданий. Несмотря на существенное сокращение

формацией понимаются:

- знания о предметах, фактах, идеях и т. д., которыми могут обмениваться люди в рамках конкретного контекста [36];
- знания относительно фактов, событий, вещей, идей и понятий, которые в определённом контексте имеют конкретный смысл [37],

таким образом, в контексте данного материала под информацией следует понимать совокупность сведений, образующих логическую схему: теоремы, научные законы, формулы, эмпирические принципы, алгоритмы, методы, законодательные и подзаконные акты и т. п.

Данные же представляют собой:

- формы представления информации, с которыми имеют дело информационные системы и их пользователи [36];
- поддающееся многократной интерпретации представление информации в формализованном виде, пригодном для передачи, связи или обработки [37],

таким образом, в контексте данного материала под данными следует понимать собой совокупность результатов наблюдений о свойствах тех или иных объектов и явлений, выраженных в объективной форме, предполагающей их многократные передачу и обработку.

Например: информацией является знание о том, что для обработки переменных выборки аналогов, имеющих распределение отличное от нормального [45], в общем случае, некорректно использовать параметрические методы [46] статистического анализа; данные в этом случае — это непосредственно сама выборка.

Иными словами, оперируя терминологией архитектуры ЭВМ [91], данные — набор значений переменных, информация — набор инструкций.

Во избежание двусмысленности в тексте данного материала эти термины приводятся именно в тех смыслах, которые описаны выше. В случае необходимости также используется более общий термин «сведения», обобщающий оба вышеуказанных понятия. В ряде случае, термины используются в соответствии с принятым значением в контексте устоявшихся словосочетаний.

числа оценщиков, имеющих право подписывать отчёты об оценке, не произошло никаких значимых изменений ни в части объёма предложения услуг, ни в части уровня цен на них. Фактически произошло лишь дальнейшее развитие уже существовавшего ранее института подписантов отчётов — оценщиков, имеющих необходимые квалификационные документы и выпускающих от своего имени отчёты, в т. ч. и те, в подготовке которых они не принимали участия. В ряде случаев подписант мог и вовсе не читать отчёт либо даже не видеть его в силу своего присутствия в другом регионе, отличном от региона деятельности компании, выпустившей отчёт. При этом, как ни странно, доход таких «специалистов» не вырос существенным образом. Всё это очевидным образом приводит к недовольству регуляторов в адрес оценочного сообщества. В таких условиях следует ожидать неизбежного дальнейшего ужесточения регулирования и усугубления положения добросовестных оценщиков и оценочных компаний. Вместе с тем было бы ошибочным считать, что виной всему являются исключительно сами оценщики и их работодатели. В существенной степени проблемы квалификации и качества работы оценщиков вызваны не их нежеланием добросовестно выполнять свою работу, а отсутствием у заказчиков интереса к серьёзной качественной оценке. Не секрет, что в большинстве случаев оценка является услугой, навязанной требованиями закона либо кредитора, не нужной самому заказчику, которого очевидно волнует не качество отчёта об оценке, а соответствие определённой в нём стоимости ожиданиям и потребностям заказчика, его договорённостям с контрагентами. В таких условиях, с одной стороны, экономика не создаёт спрос на качественную оценку, с другой — сами оценщики не предлагают экономике интересные решения и новые ценности, которые могли бы принести в отрасль дополнительные финансовые потоки.

Вторая проблема тесно связана с первой и выражается в том числе в наблюдаемом на протяжении последних примерно 10 лет падении качества отчётов об оценке и общей примитивизации работы оценщика. Суть данной проблемы можно кратко сформулировать в одной фразе: «раньше молодые оценщики спрашивали „как проанализировать данные рынка и построить модель для оценки", сейчас они задают вопрос „где взять корректировку на "X""». Установление метода корректировок в качестве доминирующего во всех случаях даже без анализа применимости других методов стало логичным итогом процесса деградации качества отчётов об оценке. При этом источником подобных корректировок чаще всего являются отнюдь не данные отрытого рынка. Как и в первом случае винить в этом только самих оценщиков было бы неправильным. В условиях работы в зачастую весьма жёстких временных рамках и за небольшое вознаграждение, оценщик часто лишён возможности провести самостоятельный анализ тех или иных свойств открытого рынка, вследствие и по причине чего вынужден использовать внешние нерыночные данные в том числе и непроверенного качества. Со временем это становится привычкой, убивающей творчество и стремление к поиску истины.

Третья проблема также неразрывно связана с двумя первыми. Отсутствие конкуренции, основанной на стремлении оказывать как можно более качественные услуги, недостаточная капитализация отрасли, выражающаяся в том числе в относительно невысоких зарплатах оценщиков, не вполне последовательное регули-

рование отрасли со стороны государства — всё это создаёт условия, при которых у оценщиков отсутствует стимул, а зачастую и возможность внедрять инновации.

Данная работа служит следующей основной цели: дать в руки оценщика инструменты, позволяющие ему просто и быстро извлекать полезные сведения из сырых данных открытых рынков, интерпретировать их, выдвигать гипотезы, выбирать среди них наиболее перспективные и в итоге получать готовые модели предсказания различных свойств объекта оценки, в том числе его стоимости. Есть некоторая надежда, что применение технологий искусственного интеллекта позволит, не увеличивая трудоёмкость, а скорее напротив, снижая её, повысить качество работы оценщика, усилить доказательную силу отчётов об оценке и в итоге позволит создать новые ценности, предлагаемые оценщиками экономике, государству, потребителям, а главное всему обществу.

Особенностью данной работы является её практическая направленность: в тексте содержатся все необходимые инструкции, формулы, описания и фрагменты программного кода либо ссылки на них, необходимые и достаточные для воспроизведения всех рассмотренных методов и их описания в отчётах об оценке.

Данная работа состоит из двух частей. Первая посвящена в большей степени теории, описанию методов, а также применению языка R [69]. Вторая имеет большую практическую направленность и содержит руководства по применению языка Python [15]. Объяснение данного факта содержится далее в разделе ССЫЛКА. В работе будут рассмотрены следующие вопросы:

a) автоматизированный сбор данных с веб-ресурсов;

b) семантический анализ текстов объявлений;

c) работа с геоданными;

d) первичная интерпретация и визуализация данных открытых рынков;

e) проверка статистических гипотез;

f) задачи классификации;

g) корреляционный анализ;

h) регрессионный анализ;

i) анализ временных рядов;

j) задачи многомерного шкалирования;

k) байесовская статистика;

l) деревья классификации;

m) случайные леса;

n) нейронные сети;

o) глубокое обучение;

p) обучение с подкреплением;

q) нечёткая логика.

Вышеприведённый перечень не является исчерпывающим и будет дорабатываться по мере развития проекта.

Данная работа основана на четырёх основополагающих принципах и предпосылках.

a) *Принцип «вся информация об активе учтена в его цене»*. Данный принцип говорит о том, что существует функциональная зависимость между ценой актива (обязательства) и его свойствами. Он тесно связан с Гипотезой эффективного рынка [75], лежащей в основе технического биржевого анализа. При этом для целей настоящей работы данная гипотеза принимается в её сильной форме эффективности [107]. С точки зрения оценщика это означает, что нет необходимости искать какие-либо данные кроме тех, которые непосредственно и объективно наблюдаются на рынке.

b) *Принцип «максимального использования релевантных наблюдаемых исходных данных и минимального использования ненаблюдаемых исходных данных»*. Данный принцип согласуется с требованиями п. 3 Международного стандарта финансовой отчётности 13 «Оценка справедливой стоимости» [119] (IFRS 13 [18]), а также, например, принципами Всемирных стандартов оценки RICS [52] (RICS Valuation — Global Standards [2]) и основывается на них. С точки зрения оценщика данный принцип означает, что лучшая практика оценки заключается в работе непосредственно с данными открытых рынков, а не чьей-либо их интерпретацией, существующей, например, в виде готовых наборов корректировок, порой весьма далёких от реальности.

c) *Принцип KISS [80]* (**k**eep **i**t **s**imple **s**tupid, вариации: **k**eep **i**t **s**hort and **s**imple, **k**eep **i**t **s**imple and **s**traightforward и т. п.), предложенный американским авиаинженером Келли Джонсоном [79], ставший официальным принципом проектирования и конструирования ВМС США с 1960 г. Данный принцип заключается в том, что при разработке той или иной системы следует использовать самое простое решение из возможных. Применительно к тематике данной работы это означает, что в тех случаях, когда автор сталкивался с проблемой выбора способа решения задачи в условиях неопределённости преимуществ и недостатков возможных вариантов, он всегда выбирал самый простой способ. Например в задаче кластеризации, выбирая между видами расстояний, автор делает выбор в пользу евклидова либо манхэттенского расстояний [76, 103].

d) *Принцип «не дай алгоритму уничтожить здравый смысл»*. Данный принцип означает необходимость самостоятельного осмысления всех результатов выполнения процедур, в т. ч. и промежуточных. Возможны ситуации, когда полученные результаты могут противоречить здравому смыслу и априорным знаниям о предметной области, которыми обладает оценщик либо пользователи его работы. Следует избегать безоговорочного доверия к результатам, выдаваемым алгоритмами. Если построенная модель противоречит априорным знаниям об окружающей реальности, то следует помнить, что другой реальности у нас нет, тогда как модель может быть скорректирована либо заменена на другую.

Все описанные этапы действий описаны таким образом, что позволяют сразу же без каких-либо дополнительных исследований воспроизвести всё, что было реализовано в данной работе. От пользователей потребуется только установить необходимые программные средства, создать свой набор данных для анализа и загрузить его в пакет. Все действия по установке и настройке описаны внутри данного руководства. Важным аспектом является то обстоятельство, что при подготовке данного исследования использовалось исключительно свободное программное обеспечение [116, 106, 115]. Таким образом, любой читатель сможет воспроизвести все описанные действия без каких-либо затрат на приобретение тех или иных программных продуктов.

От пользователей данного руководства не требуется наличие специальных познаний в области разработки программного обеспечения, software engineering и иных аспектов computer science. Некоторые понятия вроде «класс», «метод», «функция», «оператор», «регулярные выражения» и т. п. термины из сферы программирования могут встречаться в тексте руководства, однако их понимание либо непонимание пользователем не оказывает существенного влияния на восприятие материала в целом. В отдельных случаях, когда понимание термина является существенным, как например в случае с термином «переменная», в тексте руководства приводится подробное объяснение смысла такого термина, доступное для понимания неспециалиста.

Также от пользователей руководства не требуется (хотя и является желательным) глубокое понимание математической статистики, дифференциальных вычислений, линейной алгебры, комбинаторики, методов исследования операций, методов оптимизации и иных разделов математики и математической статистики, хотя и предполагается наличие таких познаний на уровне материала, включённого в школьную программу и программу технических и экономических специальностей вузов России. В тексте руководства приводится описание смысла и техники всех применённых статистических методов, математических операций и вычислений в объёме, достаточном, по мнению автора, для обеспечения доказательности при использовании методов, рассмотренных в данной работе. Автор всегда приводит ссылки на материалы, подтверждающие приведённые им описания за исключением случаев общеизвестных либо очевидных сведений. Особое внимание автор уделяет соблюдению требований к информации и данным, имеющим существенное значение

для определения стоимости объекта оценки, установленных Федеральным законом «Об оценочной деятельности в Российской Федерации» [123], а также Федеральными стандартами оценки [121].

Сведения, приведённые в настоящем руководстве, являются, по мнению автора, достаточными для обеспечения выполнения вышеуказанных требований к информации, содержащейся в отчёте об оценке. Таким образом, использование описаний процедур, приведённых в настоящем руководстве, скорее всего должно быть достаточным при использовании изложенных в нём методик в целях осуществления оценочной деятельности и составлении отчёта об оценке. Однако, автор рекомендует уточнять требования, предъявляемые к отчёту об оценке со стороны саморегулируемой организации, в которой состоит оценщик, а также со стороны заказчиков и регуляторов.

В силу свободного характера лицензии, на условиях которой распространяется данная работа, она, равно как и любая её часть, может быть скопирована, воспроизведена, переработана либо использована любым другим способом любым лицом в т. ч. и в коммерческих целях при условии распространения производных материалов на условиях такой же лицензии. Таким образом, автор рекомендует использовать тексты, приведённые в настоящем руководстве для описания выполненных оценщиком процедур.

По мнению автора, данное руководство и описанные в нём методы могут быть особенно полезны в следующих предметных областях:

- оценка и переоценка залогов и их портфелей;
- контроль за портфелями залогов со стороны регулятора банковской сферы;
- оценка объектов, подлежащих страхованию, и их портфелей со стороны страховщиков;
- оценка объектов со стороны лизинговых компаний;
- оценка больших групп активов внутри холдинговых компаний и предприятий крупного бизнеса;
- мониторинг стоимости государственного и муниципального имущества;
- оценка в целях автоматизированного налогового контроля;
- государственная кадастровая оценка;
- экспертиза отчётов об оценке, контроль за деятельностью оценщиков со стороны СРО.

Иными словами, особенная ценность применения методов искусственного интеллекта в оценке возникает там, где имеет место необходимость максимальной беспристрастности и незаинтересованности в конкретном значении стоимости.

В данном руководстве не содержатся общие выводы касательно параметров открытых рынков как таковых, не выводятся общие формулы, применимые всегда и для всех объектов оценки. Вместо этого в распоряжение пользователей предоставляется набор мощных инструментов, достаточный для моделирования ценообразования на любом открытом рынке, определения стоимости любого объекта оценки на основе его актуальных данных. В случае необходимости пользователь, применяя рассмотренные методы, может самостоятельно разработать предсказательную модель для любых рынков и объектов. Забегая вперёд, можно сказать, что при решении конкретной практической задачи применение всех описанных методов не является обязательным, а если быть точным — явно избыточным. В тексте руководства содержатся рекомендации по выбору методов на основе имеющихся свойств данных, рассматриваются сильные и слабые стороны каждого из них.

Несмотря на изначально кажущуюся сложность и громоздкость методов, при более детальном знакомстве и погружении в проблематику становится ясно, что применение предложенных реализаций методов существенно сокращает время, необходимое для выполнения расчёта относительно других методов сопоставимого качества, а сама процедура сводится к написанию и сохранению нескольких строк кода при первом применении и их вторичному многократному использованию для новых наборов данных при будущих исследованиях.

Автор выражает надежду, что данное руководство станет для кого-то первым шагом на пути изучения языков R [69] и Python [15], а также погружения в мир анализа данных, искусственного интеллекта и машинного обучения.

Глава 2.

Технологическая основа

2.1. Параметры использованного оборудования и программного обеспечения

При выполнении всех описанных в данной работе процедур, равно как и написании её текста использовалась следующая конфигурация оборудования.

Таблица 2.1.1. Параметры использованного оборудования

№	Категория	Модель (характеристика)	Источник
0	1	2	3
1	Процессор	4 ×{} Intel ® Core ™ i7-7500U CPU @ 2.70GHz	[33]
2	Память	11741076B	

При выполнении всех описанных в данной работе процедур, равно как и написании её текста использовалась следующая конфигурация программного обеспечения.

Как видно из таблиц 2.1, 2.1 для анализа данных и разработки систем поддержки принятия решений на основе искусственного интеллекта вполне достаточно оборудования, обладающего средними характеристиками, а также свободных или, по крайней мере, бесплатных программных средств.

2.2. Обоснование выбора языков R и Python в качестве средства анализа данных

2.2.1. Обоснование отказа от использования табличных процессоров в качестве средства анализа данных

На сегодняшний день очевидден факт того, что доминирующим программным продуктом, используемым в качестве средства выполнения расчётов, в среде русских оценщиков является приложение MS Excel [7]. Следом за ним идут его бесплатные аналоги LibreOffice Calc и OpenOffice Calc [17, 16], первый из которых является

Таблица 2.1.2. Параметры использованного программного обеспечения

№	Категория/наименование	Значение/версия	Источник
0	1	2	3
1	Операционная система	Kubuntu 20.04	[9]
2	KDE Plasma	5.18.5	[10]
3	KDE Frameworks	5.68.0	[10]
4	Qt	5.12.8	[60]
5	R	4.1.1 (2021-08-10) "— "Kick Things"	[69]
6	RStudio	1.4.1717	[65]
7	Git	2.25.1	[23]
8	Github Desktop	2.6.3-linux1	[25]
9	Geogebra Classic	6.0.660.0-offline	[20]
10	LaTeXDraw	4.0.3-1	[39]
11	Python	3.8.10	
12	Spyder	3.3.6	
13	PyCharm Community	2021.2.1	
14	Kate	19.12.3	

также не только бесплатным, но и свободным программным обеспечением [116, 106, 115]. В ряде случаев используется Google Sheets [27]. Не оспаривая достоинства этих продуктов, нельзя не сказать о том, что они являются универсальными средствами обработки данных общего назначения и, как любые универсальные средства, сильны своей многофункциональностью и удобством, но не шириной и глубиной проработки всех функций. Во всех вышеуказанных программных продуктах в виде готовых функций реализованы некоторые основные математические и статистические процедуры. Также само собой присутствует возможность выполнения расчётов в виде формул, собираемых вручную из простейших операторов [117]. Однако возможности этих продуктов для профессионального анализа данных абсолютно недостаточны. Во-первых, в них имеются ограничений на размер и размерность исследуемых данных. Во-вторых, в отсутствуют средства реализации многих современных методов анализа данных. Если первое ограничение не столь важно для оценщиков, редко имеющих дела с по-настоящему большими наборами данных и существенным числом переменных [99, 100] в них, второе всё же накладывает непреодолимые ограничения на пределы применимости таких программных продуктов. Например, ни одно из вышеперечисленных приложений не позволяет использовать методы непараметрической статистики [98] либо, например, решить задачи построения деревьев классификации [64] и их случайных лесов [118]. Таким образом, следует признать, что, оставаясь высококачественными универсальными средствами для базовых расчётов, вышеперечисленные приложения не могут быть использованы для профессионального анализа данных на современном уровне.

При этом их использование порой бывает необходимым на первоначальном исследования. Некоторые исходные данные, предоставляемые оценщику для обработки,

содержатся в электронных таблицах. Такие таблицы помимо полезных сведений могут содержать посторонние данные, тексты, графики и изображения. В практике автора был случай предоставления ему для анализа данных в форме электронной таблицы формата xlsx [82, 35], имеющей размер около 143 МБ, содержащей помимо подлежащей анализу числовой информации о товарах их рекламные описания в текстовом виде и фотографии, составляющие свыше 90 % размера файла. Тем не менее просмотр исходных данных средствами табличных процессоров и создание нового файла, содержащего только необходимые для анализа данные, нередко является подготовительным этапом процесса анализа. В последующих разделах будут данные практические рекомендации касательно его реализации. По мнению автора, по состоянию на 2021 год лучшим табличным процессором является LibreOffice Calc [17], превосходящий MS Excel [7] по ряду характеристик.

2.2.2. R или Python

2.2.2.1. Общие моменты

Можно с уверенностью сказать, что по состоянию на второе полугодие 2021 года доминирующими и самыми массовыми техническими средствами анализа данных, машинного обучения и разработки искусственного интеллекта[1] являются языки программирования R [69] и Python [15]. Оба они являются сверхвысокоуровневыми [104] сценарными (скриптовыми) [108] языками программирования. Высокоуровневым называется такой язык программирования, в основу которого заложена сильная абстракция, т. е. свойство описывать данные и операции над ними таким образом, при котором разработчику не требуется глубокое понимание того, как именно машина их обрабатывает и исполняет [92]. Сверхвысокоуровневым [104] языком является такой язык программирования, в котором реализована очень сильная абстракция. Иными словами, в отличие от языков программирования высокого уровня [92], в коде, разработанном на которых, описывается принцип «как нужно сделать», код, выполненный на сверхвысокоуровневых языках [104] описывает лишь принцип «что нужно сделать». Сценарным (скриптовым) [108] языком называется такой язык программирования, работа которого основана на исполнении сценариев, т. е. программ, использующих уже готовые компоненты. Таким образом, можно сделать вывод, что сверхвысокоуровневые языки лучше всего подходят для тех, кто только начинает погружаться в программирование и не обладает экспертными знаниями в вопросах архитектуры ЭВМ [91].[2]

Оба языка распространяются на условиях свободных лицензий [105] с незначительными отличиями. R распространяется на условиях лицензии GNU GPL 2 [41], Python — на условиях лицензии Python Software Foundation License [42], являющейся совместимой с GNU GPL [40]. Отличия между ними не имеют никакого практического значения для целей настоящего руководства и применения любо-

[1] Разница между этими понятиями будет описана далее в ССЫЛКА

[2] Для первичного ознакомления с вопросами архитектуры ЭВМ автор рекомендует просмотреть данный курс лекций [113].

го из этих языков в оценочной деятельности в целом. Следует лишь знать основной факт: использование этих языков является легальным и бесплатным в том числе и для коммерческих целей. Основное отличие между этими языками заключается в частности в том, что Python — язык общего назначения, широко применяемый в различных областях, тогда как R — специализированный язык статистического анализа и машинного обучения. В целом можно сказать, что задачи анализа данных могут одинаково успешно решаться средствами обоих языков. Также они оба являются Тьюринг-полными [101] языками.

Преимущества R основаны на том факте, что он изначально был разработан двумя профессиональными статистиками: Ross Ihaka [85], Robert Gentleman [83], по первым буквам имён которых он и был назван. Дальнейшее развитие языка также осуществляется прежде всего силами профессиональных математиков и статистиков, вследствие чего для R реализовано значительное количество библиотек, выполняющих практически все доступные на сегодняшнем уровне развитии науки статистические процедуры. Кроме того, можно быть уверенным в абсолютной корректности всех алгоритмов, реализованных в этих библиотеках. К тому же этот язык особенно популярен в академической среде, что означает факт того, что в случае, например, выхода какой-то статьи, описывающей новый статистический метод, можно быть уверенным, что соответствующая библиотека, реализующая этот метод выйдет в ближайшее время либо уже вышла. Кроме того, важным преимуществом R являются очень хорошо проработанные средства вывода графической интерпретации результатов анализа.

Недостатки R, как это часто бывает, следуют из его достоинств. Язык и его библиотеки поддерживаются в первую очередь силами математиков-статистиков, а не программистов, что приводит к тому, что язык относительно плохо оптимизирован с точки зрения software engineering, многие решения выглядят неочевидными и неоптимальными с точки зрения способов обращения к памяти, интерпретации в машинные команды, исполнения на процессоре. Это приводит к высокому потреблению ресурсов машины, в первую очередь памяти, медленному исполнению процедур. При этом, говоря о медленном исполнении, следует понимать относительность этой медлительности. Выполнение команды за 35 мс вместо 7 мс не замечается человеком и обычно не имеет сколько-нибудь определяющего значения. Проблемы с производительностью становятся заметны только при работе с данными большой размерности: миллионы наблюдений, тысячи переменных. В практических задачах, с которыми сталкиваются оценщики, подобная размерность данных выглядит неправдоподобной, вследствие чего можно говорить об отсутствии существенных недостатков языка R для целей применения в оценочной деятельности в целом и в целях задач, решаемых в данном руководстве, в частности. Следующей условной проблемой R является огромное количество библиотек[3] и ещё более огромное количество возможных вариантов решения задач и предлагаемых для этого методов. Даже опытный аналитик может растеряться, узнав о том, что его задача может быть ре-

[3] По состоянию на 24 августа 2021 существует 18089 официальных библиотек, содержащихся на официальной странице [61] проекта.

шена десятками способов, выбор лучшего из которых сам по себе является нетривиальной задачей. Данную особенность конечно же нельзя считать недостатком самого языка R.

Преимуществом Python является его универсальность и существенно большая распространённость. Освоение основ данного языка для целей одной предметной области может быть полезным в дальнейшем, если по каким-то причинам оценщик захочет решать с его помощью задачи иного класса. Данный язык разработан и поддерживается профессиональными программистами, что означает его относительно приемлемую оптимизацию, превосходящую R, но уступающую, например C++.

К недостаткам Python можно отнести меньшее число библиотек, содержащих статистические процедуры. Кроме того, нет такой же уверенности в безупречности их алгоритмов. При этом следует отметить, что подобные риски присутствуют лишь в новых библиотеках, реализующих экспериментальные либо экзотические статистические процедуры. Для целей оценки как правило вполне достаточно уже относительно отработанных и проверенных библиотек.

Подводя итог, можно сказать, что нет однозначного ответа, какой из вышеупомянутых языков является предпочтительным для целей анализа данных в оценке. R развивается, оптимизируется и всё больше избавляется от «детских болезней» неоптимизированности, для Python создаются новые мощные библиотеки статистического анализа. Поэтому вопрос остаётся открытым.

Следует кратко упомянуть о том, что помимо R и Python в целях анализа данных также используются вендорские программные продукты такие как SAS [32], SPSS [30], Statistica [13], Minitab [47], Stata [43], Eviews [31] и ряд других. Однако все они являются платными, при этом стоимость лицензии на самый мощный из них — SAS начинается, как правило, от нескольких десятков тысяч долларов. В остальном, кроме привычного для большинства пользователей графического интерфейса они не имеют явных преимуществ перед R и Python, предоставляя при этом даже меньше возможностей.

2.2.2.2. Современное состояние

Вышеприведённый текст, содержащийся в предыдущей секции (2.2.2.1) был написан автором в 2019 году. За прошедший период произошли некоторые изменения, требующие внимания. В настоящее время Python серьёзно опережает R по распространённости в среде аналитиков данных. Можно говорить о некотором консенсусе, согласно которому R является средством разработки и анализа данных для научных целей, тогда как Python применяется в бизнес среде. Несмотря на это, автор считает, что в целях анализа данных данные языки вполне взаимозаменяемы. Некоторые библиотеки портированы из одного из них в другой. При этом нельзя не признать, что за последние годы R существенно сдал позиции в пользу Python. В особенности это справедливо именно для российского рынка разработки систем анализа данных. Определённый пик интереса к R в России имел место в 2015–2017 годах, после чего его популярность пошла на спад. В мире пик интереса к R пришёлся на 2016–2018 годы после чего его популярность стабилизировалась. Язык продолжает активно

развивается.

В российской практике коммерческого анализа данных его заказчики, как правило, требуют реализации на Python, применение вместо него R чаще всего приходится обосновывать отдельно. Таким образом, можно говорить о том, что применение Python де факто является стандартом. Кроме того, продвижению Python во всём мире способствует позиция компаний интернет-гигантов, использующих его в своих системах машинного обучения. Следующим фактором успеха Python является его широкое распространение в теме разработки нейронных сетей, также являющееся следствием практик крупных IT-компаний. Также Python широко распространён и за пределами области анализа данных, что означает существенно большее число специалистов, владеющих им. При этом для R разработан ряд уникальных отраслевых библиотек, содержащих специфические функции. R безоговорочно лидирует в области биоинформатики, моделирования химических процессов, социологии.

При этом, R по-прежнему предоставляет существенно более широкие возможности визуализации, а также позволяет легко разрабатывать веб-интерфейсы посредством Shiny. R имеет отличный инструмент написания документации к коду в процессе разработки самого кода — R Markdown .

Подводя итоги, можно сказать о том, что современным оценщикам следует иметь навыки разработки и анализа данных с использованием обоих этих языков: R поможет применять самые свежие методы и создавать качественные понятные пользователям описания и визуализации, Python пригодится там, где требуется разработка серьёзной промышленной системы, предназначенной для многократного выполнения одинаковых задач. В целом же можно повторить основной тезис: данные языки в существенной степени взаимозаменяемы.

2.3. Система контроля версий Git

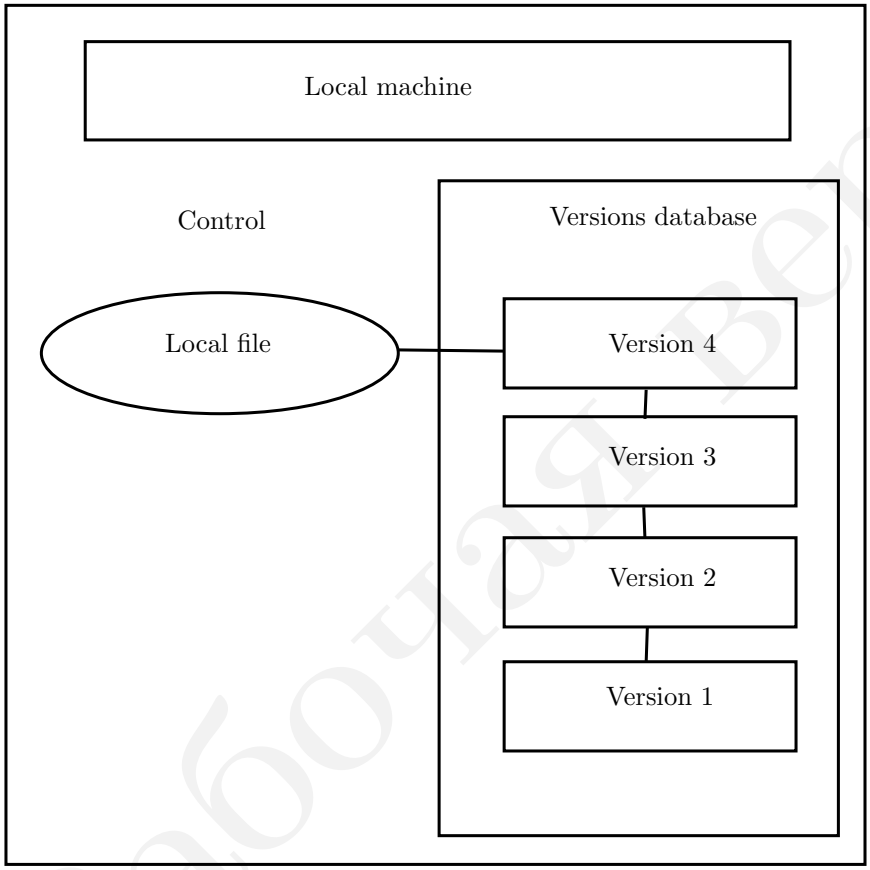
2.3.1. Общие сведения

Данный раздел не имеет отношения непосредственно к анализу данных, однако содержит сведения, полезные для комфортной работы при его осуществлении. Кроме того, использование систем контроля версий де факто является стандартом при любой серьёзной разработке, особенно в случае совместной работы над одним проектом нескольких аналитиков. Основная часть материала является пересказом видеоурока по работе с Git [114].

Система Git [23] — это одна из систем контроля версий. Система контроля версий — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии. Как правило подразумевается контроль версий файлов, содержащих исходный код программного обеспечения, хотя возможен контроль версий практически любых типов файлов [4]. Такие системы позволяют не только хранить версии файлов, но и содержат всю историю их изменения, позволяя отслеживать пошаговое изменение каждого бита файла. Это бывает особенно полезно в тех случаях, когда необходимо иметь возможность

«откатить» изменения в случае наличия в них ошибок либо тогда, когда над одним и тем же проектом работает несколько разработчиков либо их команд. Конечно же можно просто создавать полные копии всех файлов проекта. Однако данный способ полезен лишь для создания бэкапов на случай каких-то аварийных ситуаций. В обычной работе он, как минимум, неудобен, а, как максимум, просто не способен обеспечить пошаговое отслеживание изменений файлов и тем более слияние результатов нескольких команд, параллельно работающих над одними и теми же файлами. Для решения данной проблемы были разработаны локальные системы контроля версий, содержащие базу данных всех изменений в файлах, примерная схема организации которых показана на рисунке 2.3.1.

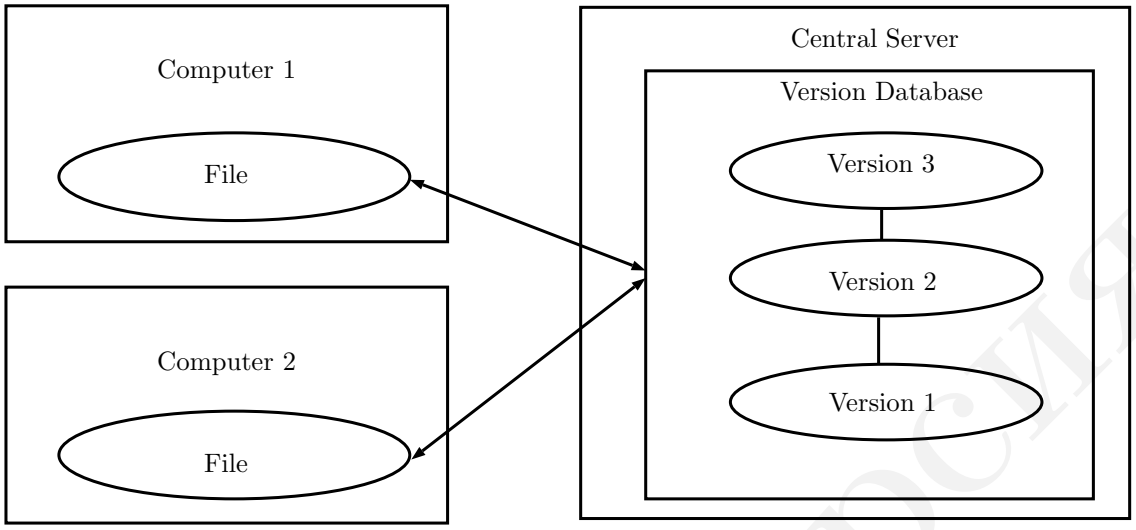
Рис. 2.3.1. Локальная система контроля версий



Современные системы контроля версия бывают централизованными и распределёнными. Первые устроены таким образом, что вся история изменений файлов хранится на центральном сервере, на который пользователи отправляют свои изменения, и с которого они их получают. Общая схема работы централизованной системы контроля версий приведена на рисунке 2.3.2 на следующей странице. Недостатком такой системы являет её зависимость от работы центрального сервера. В случае его остановки пользователи не смогут обрабатывать изменения, принимать и отправлять их. Также существует риск полной потери всей истории в случае оконча-

тельного отказа сервера.

Рис. 2.3.2. Схема работы централизованной системы контроля версий

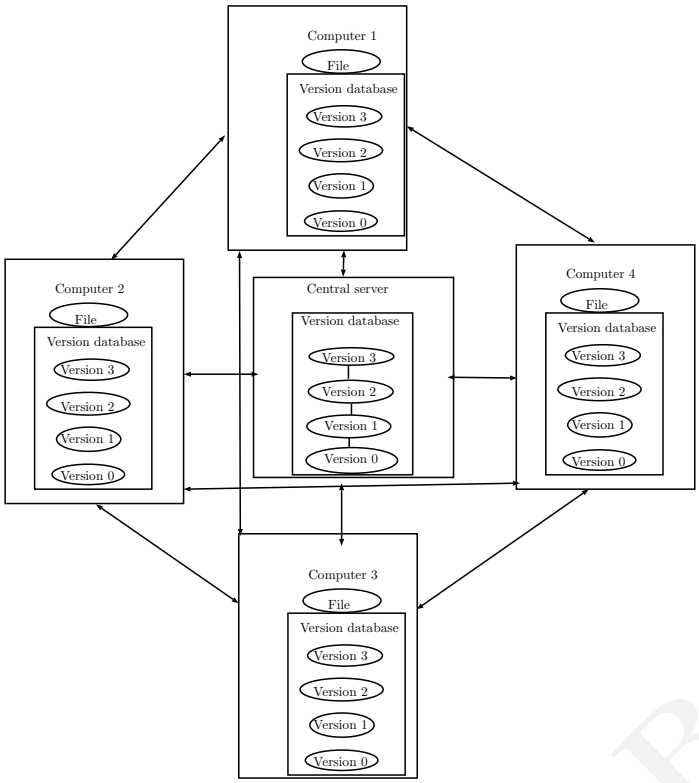


Распределённые системы контроля версия лишены данного недостатка, поскольку у каждого пользователя хранится полная история изменений. В связи с этим каждый пользователь может продолжать работать с системой контроля при отсутствии связи с сервером. После восстановления работоспособности последнего, пользователь сможет синхронизировать свою историю изменений с другими разработчиками. Даже в случае полного отказа сервера команда сможет просто перевести хранение на другой и продолжить работу в прежнем режиме. Общая схема работы распределённой системы приведена на рисунке 2.3.3.

Особенностью работы системы Git является заложенный в ней принцип работы. В отличие от некоторых других систем контроля версий, принцип которых основан на хранении исходного файла и списка изменений к нему, Git хранит состояние каждого файла после его сохранения, создавая его «снимок». В терминологии Git каждый такой снимок называется commit. При этом создаются ссылки на каждый из файлов. В случае, если при создании нового commit Git обнаруживает, что какие-то файлы не были изменены, система не включает сами файлы в новый commit, а лишь указывает ссылку на последнее актуальное состояние файла из предыдущего commit, обеспечивая таким образом эффективность дискового пространства. При этом каждый commit в целом ссылается на предыдущий, являющийся для него родительским. На рисунке 2.3.4 на с. 35 показана общая схема работы системы Git. Линиями со сплошным заполнение показана передача нового состояния файла, возникшего в результате внесения в него изменений, прерывистым — передача ссылки на состояние файла, не подвергавшегося изменениям, из прежнего commit. На момент времени 0 (initial commit) все файлы находились в состоянии 0. Затем в файлы B и C были внесены изменения, тогда как файл A остался в прежнем состоянии. В процессе создания commit № 1 Git сделал снимок состояния файлов B1 и C1, а также создал ссылку на состояние файла A0. Далее изменения были внесены в файл

Рис. 2.3.3. Схема работы распределённой системы контроля версий

.



В. В процессе создания commit №2 Git сохранил состояние файла B2, а также создал ссылки на состояния файлов A0 и C1 в предыдущем commit №1. Затем были внесены изменения во все три файла, в результате чего на этапе создания commit №3 Git сделал снимок состояний всех трёх файлов.

Внимательный читатель скорее всего обратил внимание на третий тип линий — пунктир, которому соответствует подпись «hash». Чтобы понять, каким образом в Git реализуется целостность версий, необходимо обратиться к понятию хеш-функции [8, 109].

2.3.2. Хеш-функции

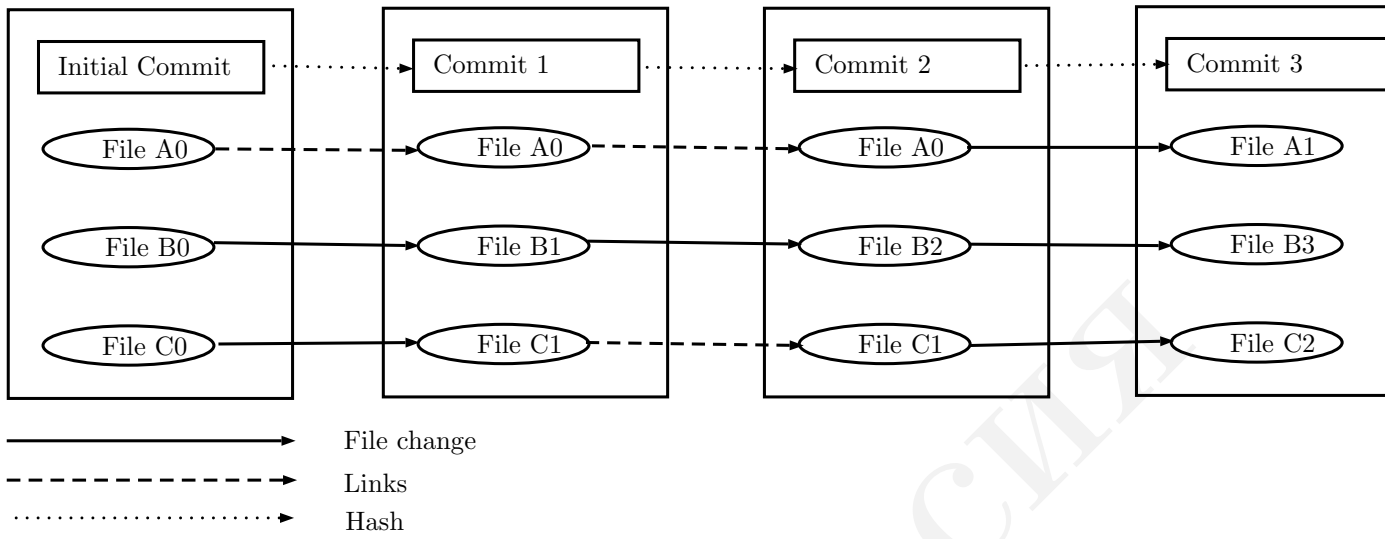
Приведём основные определения.

Хеш функция (функция свёртки) — функция, представляющая собой детерминированный математический алгоритм [93], осуществляющая преобразование данных произвольной длины в результирующую битовую строку фиксированной длины.

Хеширование — преобразование, осуществляемое хеш-функцией.

Сообщение (ключ, входной массив) — исходные данные.

Рис. 2.3.4. Общая схема работы Git



Хеш (хеш-сумма, хеш-код, сводка сообщения) — результат хеширования.

Согласно Принципу Дирихле [102], между хешем и сообщением в общем отсутствует однозначное соответствие. При этом, число возможных значений хеша меньше числа возможных значений сообщения. Ситуация, при которой применение одной и той же хеш-функции к двум различным сообщениям приводит к одинаковому значению хеша, называется «коллизией хеш функции» [96]. Т. е. коллизия имеет место тогда, когда $H(x) = H(y)$.

Теоретическая «идеальная» хеш-функция отвечает следующим требованиям:

a) является детерминированной, то есть её применение к одному и тому же сообщению приводит к одному и тому же значению хеша любое число раз;

b) значение хеша быстро вычисляется для любого сообщения;

c) зная значение хеша, невозможно определить значение сообщения;

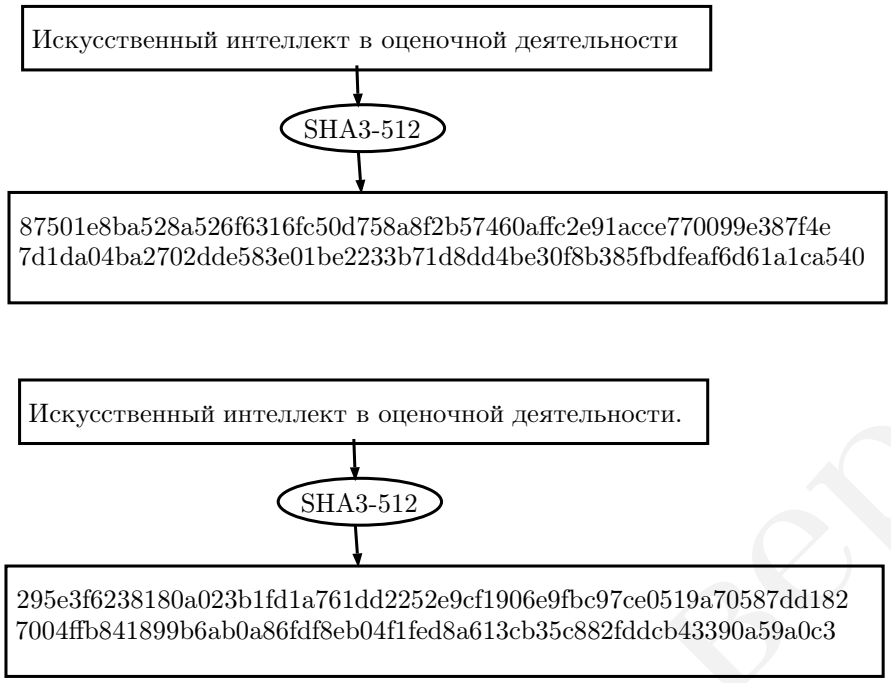
d) невозможно найти такие два разных сообщения, применение хеширование к которым приводило бы к одинаковому значению хеша (т. е. идеальная хеш-функция исключает возможность возникновения коллизии);

e) любое изменение сообщения (вплоть до изменения значения одного бита) изменяет хеш настолько сильно, что новое и старое значения выглядят никак не связанными друг с другом.

Как правило, название хеш-функции содержит значение длины результирующей битовой строки. Например хеш-функция SHA3-512 [87] возвращает строку длиной в 512 бит. Воспользуемся одним [62] из онлайн-сервисов вычисления хеша и посчитаем его значение для названия данной книги. Как видно на рисунке 2.3.5 на следующей странице, результатом вычисления хеш-функции является строка длиной

в 512 бит, содержащая 128 шестнадцатеричных чисел. При этом, можно наблюдать, что добавление точки в конце предложения полностью меняет значение хеша.

Рис. 2.3.5. Пример вычисления хеша



Длина хеша в битах определяет максимальное количество сообщений, для которых может быть вычислен уникальный хеш. Расчёт осуществляется по формуле.

$$2^{n} \tag{2.3.1}$$

, где n — длина строки в битах.

Так, для функции SHA3-512 число сообщений, имеющих уникальный хеш составляет: $2^{512} \sim 1.340781 \times 10^{154}$. Таким образом, можно говорить о том, что современные хеш-функции способны генерировать уникальный хеш для сообщений любой длины.

Таким образом, Git в процессе создания нового commit сначала вычисляет его хеш-сумму, а затем фиксирует состояние. При этом в каждом commit присутствует ссылка на предыдущий, также имеющий свою хеш-сумму. Таким образом, обеспечивается целостность истории изменений, поскольку значение хеш-суммы каждого последующего commit вычисляется на основе сообщения, содержащего в т. ч. свою хеш-сумму. В этом случае любая модификация содержимого данных, образующих любой commit, неизбежно приведёт к изменению всех последующих хешей, что не останется незамеченным.

2.3.3. Начало работы с Git и основные команды

Для того, чтобы начать работать с Git прежде всего его конечно же следует установить. Как правило, с этим не возникает никаких сложностей. Однако всё же вопросы установки Git кратко рассмотрены в подразделе 2.4.1 Git 66–67.

В данном подразделе преимущественно рассматриваются аспекты работы с ним через командную строку. Данный выбор обусловлен тем обстоятельством, что существует множество графических интерфейсов для работы с Git, которые активно развиваются, меняют дизайн и расширяют функционал. Кроме того, появляются новые продукты. Среди такого разнообразия всегда можно выбрать какой-то наиболее близкий для себя вариант. Таким образом, автор не видит смысла останавливаться на разборе какого-то конкретного графического интерфейса. Более важной задачей является изложение сути и основных принципов работы, понимание которых обеспечит успешную работы с Git безотносительно конкретных программных средств. Кроме того, следует отметить, что практически все современные IDE [95] имеют свои средства и интерфейс для работы с Git. В дальнейшем в главах, посвящённых непосредственно применению R и Python, будут рассмотрены вопросы использования Git средствами RStudio, Spyder и PyCharm.

В данном подразделе описывается работа с Git через командную строку в операционной системе Kubuntu. Большая часть изложенного применима для любой операционной системы. Для начала работы с Git откроем терминал и выполним три основные настройки, а именно укажем:

- имя пользователя;
- адрес электронной почты;
- текстовый редактор по умолчанию.

Для конфигурации Git существует специальная утилита *git config*, имеющая три уровня глобальности настроек:

- ```
  $ git config --system
  ```
  — системный уровень: затрагивает все репозитории всех пользователей системы;
- ```
  $ git config --global
  ```
  — глобальный уровень: затрагивает все репозитории конкретного пользователя системы;
- ```
  $ git config --local
  ```
  — локальный уровень: затрагивает конкретный репозиторий;
  ```

Представим, что необходимо задать общие настройки конкретного пользователя, т. е. использовать уровень global, что, может быть актуально, например, при использовании рабочего компьютера. Сделаем следующие настройки:

```
$ git config --global user.name "First.Second"
$ git config --global user.email user-adress@host.com
$ git config --global core.editor "kate"
```

— мы задали имя пользователя, адрес его электронной почты, отображаемые при выполнении commit, а также указали текстовый редактор по умолчанию. В данном случае был указан редактор Kate. Естественно можно указать любой другой удобный редактор. В случае использования операционной системы Windows необходимо указывать полный путь до исполняемого файла (имеет расширение .exe) текстового редактора, а также а. Например, в случае использования 64-х разрядной Windows и редактора Notepad++ [55] команда может выглядеть так:

```
$ git config --global core.editor "'C:\Program␣Files\Notepad
   \notepad.exe'␣-multiInst␣-notabbar␣-nosession␣-noPlugin"
```

— перечень команд для различных операционных систем и текстовых редакторов содержится на соответствующей странице сайта Git [23].

Для начала создадим тестовый каталог, с которым и будем работать в дальнейшем при обучении работе с Git. Зайдём в папку, в которой хотим создать каталог и запустим терминал в ней. После чего введём команду:

```
$ mkdir git-lesson
```

— мы только что создали новый каталог средствами командной строки.

Затем введём команду:

```
$ cd git-lesson
```

— переходим в только что созданный каталог.

Для просмотра содержимого каталога используем следующую команду:

```
$ ls -la
```

— собственно самой командой является `ls`, а «`-la`» представляет собой её аргументы: «`-l`» — отвечает за отображение файлов и подкаталогов списком, а «-a» — за отображение скрытых файлов и подкаталогов.

Для создания репозитория введём команду:

```
$ git init
```

— Git ассоциирует текущую папку с новым репозиторием.

В случае, если всё прошло хорошо, терминал возвратит следующее сообщение:

```
> Initialized empty Git repository in /home/.../git-lesson/.
   git/
```

Теперь ещё раз введём:

```
$ ls -la
```

— следует обратить внимание на то, что появилась папка .git, в которой и будет храниться вся история версий проекта, содержащегося в папке git-lesson.

Создадим первый файл внутри папки:

```
$ touch file1.py
```

— расширение указывает на то, что это файл языка Python.

Система Git уже должна была отследить наличие изменения состояния проекта, произошедшее вследствие создания нового файла. Для проверки изменений состояния используем команду:

```
$ git log
```

— и получим сообщение следующего содержания:

```
> fatal: your current branch 'master' does not have any
   commits yet
```

— дело в том, что в истории изменений по-прежнему нет никаких записей.

Для получения дополнительных сведений используем команду:

```
$ git status
```

— терминал возвратит следующее сообщение:

```
> On branch master

No commits yet

Untracked files:
  (use "git␣add␣<file>..." to include in what will be
     committed)
        file1.py

nothing added to commit but untracked files present (use "
   git␣add" to track)
```

— как видно, Git сообщает о том, что файл file1.py не отслеживается, кроме того, как следует из последней части сообщения терминала, в настоящее время вообще не фиксируются никакие изменения, поскольку ничего не было добавлено в лист отслеживания. При этом сам Git предлагает использовать команду `git add` для добавления файлов в него. Прежде чем сделать это, необходимо разобраться в том, в каких состояниях, с точки зрения Git, могут в принципе находиться файлы.

Все файлы, находящиеся в рабочем каталоге, могут иметь один из следующих статусов:

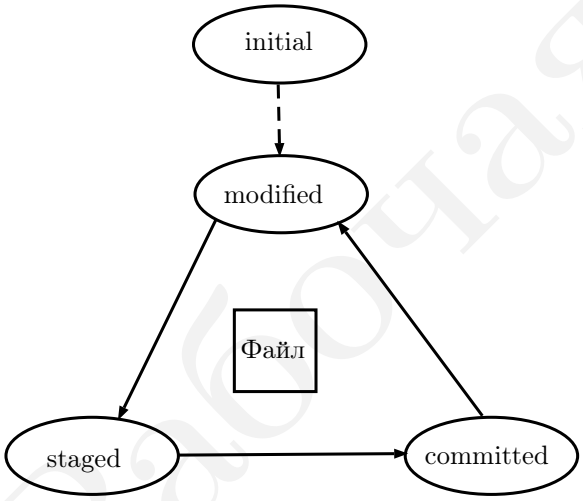
- tracked — отслеживаемые, т. е. находящиеся под версионным контролем;
- untracked — не отслеживаемые, т. е. не находящиеся под версионным контролем.

Ко второй категории, как правило, относятся временные файлы, например логи, хранение которых в репозитории нецелесообразно. Файлы первой категории могут находиться в одной из следующих состояний:

- initial — начальное состояние файла, в котором он находился в момент включения его в лист отслеживания, т. е. сообщения ему статуса tracked.
- modified — состояние файла после внесения в него изменений и его сохранения;
- staged — промежуточное состояние файла, в котором он находится после передачи его состояния Git, но до формирования последним его снимка.
- committed — состояние файла, зафиксированное Git, и представляющее его версию, к которой впоследствии будет возможно вернуться.

Соответственно после внесения новых изменений файл, находящийся в состоянии committed, переходит в состояние modified, после чего возможен новый цикл преобразований его статуса. Схема изменений состояния файлов приведена на рисунке 2.3.6.

Рис. 2.3.6. Схема состояний файлов в системе Git



Для перевода файла из состояния modified в состояние staged следует использовать команду

```
$ git add <file.name1> <file.name2>
```

— данная процедура также называется добавлением файла в индекс. Индекс — область памяти, в которой находятся файлы, подготовленные для включения в commit.

Далее для выполнения процедуры commit даётся команда

```
$ git commit -m "message"
```

— аргумент `-m` и следующее за ним сообщение служат для задания краткого описания того, какие изменения были внесены. Рекомендуется давать содержательные комментарии, позволяющие понять смысл изменений.

Как видно, не обязательно совершать процедуру commit сразу в отношении всех файлов, находящихся в состоянии modified. Существует возможность группировать их и, посредством перевода конкретных файлов в состояние staged, формировать группы файлов, чьё состояние подлежит фиксации.

Добавим файл `file.py` в индекс.

```
$ git add file1.py
```

Далее снова проверим статус:

```
$ git status
```

— на этот раз терминал возвратит новое сообщение:

```
> On branch master

No commits yet

Changes to be committed:
  (use "git␣rm␣--cached␣<file>..." to unstage)
        new file:   file1.py
```

Как можно видеть, теперь Git «видит» файл file1.py и готов сделать «снимок» нового состояния репозитория. Для выполнения процедуры commit введём команду:

```
$ git commit -m "First␣commit"
```

— мы только что сделали первый commit, т. е. зафиксировали состояние репозитория. Терминал возвратит следующее сообщение:

```
> [master (root-commit) 1306b16] First commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 file1.py
```

Теперь повторим ранее уже использованную команду:

```
$ git log
```

— терминал в отличие от первого раза, когда мы наблюдали сообщение о невозможности выведения сведений о событиях в репозитории, на этот раз возвращает осмысленное сообщение:

```
> commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1 (HEAD ->
   master) Author: Kirill Murashev <kirill.murashev@gmail.
   com>
```

```
Date:   Tue Aug 31 19:03:49 2021 +0200
    First commit
```

— можно увидеть хеш-сумму данного commit, его автора, а также время создания commit и сопроводительное сообщение к нему. Для получения более детальных сведений можно использовать команду `git show`, сообщив ей в качестве аргумента хеш-сумму интересующего commit. Сделаем это, скопировав и вставив значение хеш-суммы:[4]

```
$ git show 1306b16f5fe40ccf8b141d716d9313df8e1983a1
```

— в качестве аргумента команды в данном случае была использована хеш-сумма. Терминал возвратит сообщение с данными об интересующем commit:

```
> commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1 (HEAD ->
   master)
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 19:03:49 2021 +0300

    First commit

diff --git a/file1.py b/file1.py
new file mode 100644
index 0000000..e69de29
```

В дополнение к уже имеющимся данным приводятся сведения о том, какие именное изменения имели место. В данном случае видно, что имело место добавление в репозиторий нового файла.

Примерно такие же сведения можно получить в случае использования команды `git log` с аргументом `-p`.

```
$ git log -p

> commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1 (HEAD ->
   master) Author: Kirill Murashev <kirill.murashev@gmail.
   com> Date:   Tue Aug 31 19:03:49 2021 +0300

    First commit

diff --git a/file1.py b/file1.py
new file mode 100644
index 0000000..e69de29
```

— в данном случае сообщения вообще идентичны.

[4] Для копирования и вставки в окне терминала следует использовать сочетания клавиш ctrl+shift+c, ctrl+shift+v соответственно.

Рассмотрим ещё одну полезную команду `git restore`. Данная команда возвращает состояние файла к тому состоянию, которое было зафиксировано при создании последнего commit. Рассмотрим пример. Откроем файл file1.py в редакторе Kate[5] непосредственно из терминала:

```
$ kate file.py
```

— далее напишем в нём любой текст и сохраним файл. После чего проверим его статус с помощью уже известной команды `git status`:

```
$ git status

> On branch master
Changes not staged for commit:
   (use "git␣add␣<file>..." to update what will be committed
      )
  (use "git␣restore␣<file>..." to discard changes in working
      directory)
        modified:   file1.py

no changes added to commit (use "git␣add" and/or "git␣commit
   ␣-a")
```

— как видим, Git обнаружил изменение файла. Теперь введём команду:

```
$ git restore file.py
```

— файл, возвращён в состояние, в котором он находился на момент создания последнего commit, т. е. снова является пустым, в чём легко убедиться, открыв его.

Следующей рассматриваемой командой будет `git diff`. Данная команда позволят понять, какие именно изменения были внесены в файл. Вновь откроем файл file1.py в текстовом редакторе. Введём в него текст, например «Liberte, egalite, fraternite». После чего сохраним файл. Выполним команду `git diff` и посмотрим на результат.

```
$git diff

> diff --git a/file1.py b/file1.py
index e69de29..72d6a2a 100644
--- a/file1.py
+++ b/file1.py
@@ -0,0 +1 @@
+Liberte, egalite, fraternite
```

— в нижней части сообщения терминала после символа «+» мы видим добавленный в файл текст. Git всегда отображает добавленный текст после знака «+», а удалённый после знака «-». Проверим статус файла:

[5] Естественно редактор может быть любой

```
$ git status

> On branch master
Changes not staged for commit:
  (use "git␣add␣<file>..." to update what will be committed)
 (use "git␣restore␣<file>..." to discard changes in working
    directory)
        modified:   file1.py

no changes added to commit (use "git␣add" and/or "git␣commit
   ␣-a")
```

— Git зафиксировал изменения файла. Теперь добавим файл в индекс, т. е. изменим его состояние на staged:

```
$ git add file1.py
```

— далее ещё раз проверим статус файла:

```
$ git status

> On branch master
Changes to be committed:
  (use "git␣restore␣--staged␣<file>..." to unstage)
        modified:   file1.py
```

— Git перевёл файл в состояние staged. Для того, чтобы ещё раз просмотреть изменения в файле, находящемся в состоянии staged можно использовать ту же команду `git diff`, при условии сообщения ей аргумента `--staged`, без которого она не сможет отобразить изменения, поскольку они уже были включены в индекс.

```
$ git diff --staged

> diff --git a/file1.py b/file1.py
index e69de29..d77d790 100644
--- a/file1.py
+++ b/file1.py
@@ -0,0 +1 @@
+Liberte, egalite, fraternite
```

Выполним commit:

```
$ git commit -m "Second␣commit"
```

— терминал возвратит сообщение:

```
> [master 700a993] Second commit
 1 file changed, 1 insertion(+)
```

— посмотрим на историю изменений:

```
$ git log

> commit 700a993db7c5f682c33a087cb882728adc485198 (HEAD ->
   master)
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 20:51:06 2021 +0200

    Second commit

commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 19:03:49 2021 +0200

    First commit
```

— можно наблюдать сведения о двух выполненных commit.

В случае использования той же команды с аргументом `-p` можно увидеть всю историю конкретных изменений.

```
$ git log -p
> commit 700a993db7c5f682c33a087cb882728adc485198 (HEAD ->
   master)
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 20:51:06 2021 +0300

    Second commit

diff --git a/file1.py b/file1.py
index e69de29..d77d790 100644
--- a/file1.py
+++ b/file1.py
@@ -0,0 +1 @@
+Liberte, egalite, fraternite

commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 19:03:49 2021 +0300
    First commit
diff --git a/file1.py b/file1.py
new file mode 100644
index 0000000..e69de29
```

Существует упрощённый способ передачи Git сведений для совершения commit. Вместо последовательного ввода команд `git add` с указанием перечня файлов и `git`

`commit` можно использовать единую команду `git commit` с аргументами `-am`. Второй аргумент, как уже было сказано ранее, необходим для формирования сообщения, сопровождающего commit. Первый же заменяет собой предварительное использование команды `git add`, указывая Git на необходимость включения в индекс всех отслеживаемых файлов, т. е. имеющих статус tracked. Внесём любые изменения в файл file1.py. Проверим наличие изменений:

```
$ git status

> On branch master
Changes not staged for commit:
  (use "git␣add␣<file>..." to update what will be committed)
  use "git␣restore␣<file>..." to discard changes in working
     directory)
        modified:   file1.py

no changes added to commit (use "git␣add" and/or "git␣commit
   ␣-a")
```

— после чего выполним добавление в индекс и commit одной командой.

```
$ git commit -am "Third␣commit"
> [master fbff919] Third commit
 1 file changed, 1 insertion(+)
```

— проверим историю:

```
$ git log -p

> commit fbff919fab14ab6d41c993d3b86253c41037e075 (HEAD ->
   master)
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 21:25:45 2021 +0300

    Third commit

diff --git a/file1.py b/file1.py
index d77d790..bf6409f 100644
--- a/file1.py
+++ b/file1.py @@ -1 +1,2 @@
 Liberte, egalite, fraternite
+Жизнь, свобода, собственность

commit 700a993db7c5f682c33a087cb882728adc485198
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 20:51:06 2021 +0300
```

```
    Second commit

diff --git a/file1.py b/file1.py
index e69de29..d77d790 100644
--- a/file1.py
+++ b/file1.py
@@ -0,0 +1 @@
+Liberte, egalite, fraternite

commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 19:03:49 2021 +0300

    First commit

diff --git a/file1.py b/file1.py
new file mode 100644
index 0000000..e69de29
```

— можно наблюдать уже три commit.

Следующей полезной командой является `git mv`. Данная команда позволяет, в частности, переименовывать либо перемещать файлы. При этом её выполнение автоматически переводит файл в состояние staged, минуя состояние modified. Выполним переименование:

```
$ git mv file1.py file-1.py
```

— затем проверим состояние:

```
$ git status

> On branch master
Changes to be committed:
  (use "git␣restore␣--staged␣<file>..." to unstage)
        renamed:    file1.py -> file-1.py
```

— как можно увидеть, файл с новым именем готов к commit. Выполним commit.

```
$ git commit -m "Fourth commit"

> [master 284073c] Fourth commit
 1 file changed, 0 insertions(+), 0 deletions(-)
 rename file1.py => file-1.py (100%)
```

— изменения файла зафиксированы.

Следующей заслуживающей внимания командой является `git rm`. Данная команда удаляет файл.

```
$ git rm file-1.py
```

— проверим выполнение операции:

```
$ git status

> On branch master
Changes to be committed:
  (use "git␣restore␣--staged␣<file>..." to unstage)
        deleted:    file-1.py
```

— как видно из сообщения Git в терминале, существует возможность восстановить удалённый файл в том состоянии, которое было зафиксировано при выполнении последнего commit. Выполним команду для восстановления файла:

```
$ git restore --staged file-1.py
```

— затем проверим его состояние:

```
$ git status

> On branch master
Changes not staged for commit:
  (use "git␣add/rm␣<file>..." to update what will be
     committed)
  (use "git␣restore␣<file>..." to discard changes in working
      directory)
        deleted:    file-1.py

no changes added to commit (use "git␣add" and/or "git␣commit
   ␣-a")
```

— как следует из сообщения Git, файл file-1.py больше не находится в индексе, для его возвращения туда необходимо выполнить команду `git restore` без указания каких-либо аргументов.

```
$ git restore file-1.py
```

— ещё раз проверим состояние:

```
$ git status

> On branch master nothing to commit, working tree clean
```

— файл снова включён в индекс, его состояние соответствует состоянию, зафиксированному при выполнении последнего commit. Сам файл при этом вновь присутствует в каталоге.

Команда `git rm` также может быть использована для передачи файлу статуса untracked без его удаления из каталога. Для этого ей необходимо сообщить аргумент `--cached`.

```
$  git rm --cached file-1.py

>rm 'file-1.py'
```

— файл был исключён из индекса, а также из списка отслеживания, но при этом остался в каталоге, в чём можно легко убедиться:

```
$ git status
> On branch master
Changes to be committed:
  (use "git␣restore␣--staged␣<file>..." to unstage)
        deleted:    file-1.py

Untracked files:
  (use "git␣add␣<file>..." to include in what will be
     committed)
        file-1.py
```

— есть изменения, доступные для commit, а также в каталоге присутствует неотслеживаемый файл (статус untracked).

```
$ ls -la
> total 0\
drwx------ 1 user.name root  0 jaan   1  1970 .
drwx------ 1 user.name root  0 jaan   1  1970 ..
-rwx------ 1 user.name root 84 sept   1 19:08 file-1.py
drwx------ 1 user.name root  0 jaan   1  1970 .git
```

— файл присутствует в каталоге.

Выполним commit:

```
$ git commit -m "Fifth␣commit"
> [master 7abee55] Fifth commit
 1 file changed, 2 deletions(-)
 delete mode 100644 file-1.py
```

— далее посмотрим историю изменений:

```
$ git log

> commit 7abee55d2631cf7cf2e94e58f30f36b2be807948 (HEAD ->
   master)
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Wed Sep 1 19:52:04 2021 +0300
```

```
    Fifth commit

commit 284073c521af8b73e16f324698f24040e4b9ee7e
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Wed Sep 1 18:16:46 2021 +0300

    Fourth commit

commit fbff919fab14ab6d41c993d3b86253c41037e075
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 21:25:45 2021 +0300

    Third commit

commit 700a993db7c5f682c33a087cb882728adc485198
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 20:51:06 2021 +0300

    Second commit

commit 1306b16f5fe40ccf8b141d716d9313df8e1983a1
Author: Kirill Murashev <kirill.murashev@gmail.com>
Date:   Tue Aug 31 19:03:49 2021 +0300

    First commit
```

— проверим наличие файла в каталоге:

```
$ ls -la
> total 0
drwx------ 1 user.name root  0 jaan   1  1970 .
drwx------ 1 user.name root  0 jaan   1  1970 ..
-rwx------ 1 user.name root 84 sept   1 19:08 file-1.py
drwx------ 1 user.name root  0 jaan   1  1970 .git
```

— а также его статус:

```
$ git status
> On branch master
Untracked files:
  (use "git␣add␣<file>..." to include in what will be
     committed)
        file-1.py

nothing added to commit but untracked files present (use "
   git␣add" to track)
```

```
```

— файл присутствует в каталоге и имеет статус untracked.

Вернём файл в индекс.

```
$ git add file-1.py
```

— файл вновь имеет статус tracked.

2.3.4. Исключение файлов из списка отслеживания

В процессе разработки нередко возникают файлы, отслеживание которых скорее всего является нецелесообразным, например файлы, содержащие логи. При этом их постоянное присутствие в списке файлов, имеющих статус untracked, осложняет работы и также является нежелательным. В связи с этим существует механизм исключения ряда файлов или подкаталогов из под всей системы версионирования, называемый *gitignore*.

Выполним ряд процедур. До этого все действия выполнялись путём последовательного ввода команд. В данном случае будет показано, как можно использовать заготовленные скрипты. Использование скриптов является очень удобным тогда, когда существует необходимость многократного ввода длинной последовательности команд. В рассматриваемом примере будет рассмотрена последовательность всего из пяти команд. Для создания скрипта необходимо написать его текст в текстовом редакторе, сохранить файл с расширением txt (например script1.txt), после чего запустить терминал в каталоге с файлом и указать системе на то, что данный файл является исполняемым, т. е. передать ему права execute. Напишем скрипт:

```
#создаём подкаталог
mkdir log
#переходим в новый подкаталог
cd log/
#создаём файл
touch log.txt
#возвращаемся в каталог верхнего уровня
cd ..
#проверяем статус
git status
```

— смысл того, что выполняет команда раскрыт в комментарии, предшествующем ей. Следует обратить внимание на то, что команды, передаваемые терминалу пишутся на языке Bash [70], в котором игнорируется всё, что написано в строке после символа «#». Передадим файлу права execute путём ввода команд в терминала, запущенном из каталога, содержащего файл. Можно использовать любую (двоичную либо символическую) запись:

```
$ chmod u+x script
```

— либо:

```
$ chmod 744 script
```

— для проверки наличия прав в системе Kubuntu и многих других можно использовать команду:

```
$ ls -l script1
```

— в случае наличия прав execute терминал возвратит ответ, содержащий имя файла, выделенное зелёным цветом.

Теперь следует вернуться в окно терминала, запущенное в каталоге изучаемого репозитория после чего просто ввести нём полный путь до созданного скрипта:

```
$ ~/.../Scripts/script1
```

— в случае правильных действий терминал возвратит сообщение:

```
> On branch maste
r Changes to be committed:
  (use "git␣restore␣--staged␣<file>..." to unstage)
        new file:   file-1.py

Untracked files:
  (use "git␣add␣<file>..." to include in what will be
     committed)
        log/
```

В данном случае автор использовал заготовленный bash скрипт. Аналогичного результата можно добиться путём простого последовательного ввода команд. Подробнее о запуске скриптов в операционных системах, основанных на ядре Linux, можно прочитать, например здесь [112]. Возвращаясь к теме Git, отметим, что в каталоге появилась неотслеживаемая папка log. Создадим файл с именем .gitignore:

```
$ kate .gitignore
```

— при этом сразу же откроется окно текстового редактора. Следует сделать небольшое отступление и сказать о том, что состав файлов и папок, подлежащих исключению из списка, подлежащего версионированию, в существенной степени зависит от используемого языка программирования. В дальнейшем будут рассмотрены вопросы автоматизации создания файла .gitignore. Сейчас же кратко рассмотрим заготовленные файлы для языков Python и R. Ниже приводится примерное содержание файла .gitignore, предназначенного для репозитория, содержащего код на языке Python:

```
1 # Byte-compiled / optimized / DLL files
2 __pycache__/
3 *.py[cod]
4 *$py.class
```

```
# C extensions
*.so

# Distribution / packaging
.Python
build/
develop-eggs/
dist/
downloads/
eggs/
.eggs/
lib/
lib64/
parts/
sdist/
var/
wheels/
share/
python-wheels/
*.egg-info/
.installed.cfg
*.egg MANIFEST

# PyInstaller
#  Usually these files are written by a python script from a template
#  before PyInstaller builds the exe, so as to inject date/other infos into it.
*.manifest
*.spec

# Installer logs
pip-log.txt
pip-delete-this-directory.txt

# Unit test / coverage reports
htmlcov/
.tox/
.nox/
.coverage
.coverage.*
.cache nosetests.xml
coverage.xml
```

```
*.cover
*.py,cover
.hypothesis/
.pytest_cache/
cover/

# Translations
*.mo
*.pot

# Django stuff:
*.log
local_settings.py
db.sqlite3
db.sqlite3-journal

# Flask stuff:
instance/
.webassets-cache

# Scrapy stuff:
.scrapy

# Sphinx documentation
docs/_build/

# PyBuilder
.pybuilder/
target/

# Jupyter Notebook
.ipynb_checkpoints

# IPython
profile_default/
ipython_config.py

# pyenv
#   For a library or package, you might want to ignore these files since the code is
#   intended to run in multiple environments; otherwise, check them in:
# .python-version
# pipenv
```

```
#   According to pypa/pipenv#598, it is recommended to
    include Pipfile.lock in version control.
#   However, in case of collaboration, if having platform-
    specific dependencies or dependencies
#   having no cross-platform support, pipenv may install
    dependencies that don't work, or not
#   install all needed dependencies.
#Pipfile.lock
# PEP 582; used by e.g. github.com/David-OConnor/pyflow
__pypackages__/

# Celery stuff
celerybeat-schedule
celerybeat.pid

# SageMath parsed files
*.sage.py

# Environments
.env
.venv
env/
venv/
ENV/
env.bak/
venv.bak/

# Spyder project settings
.spyderproject
.spyproject

# Rope project settings
.ropeproject

# mkdocs documentation
/site

# mypy
.mypy_cache/
.dmypy.json dmypy.json

# Pyre type checker
.pyre/

```

```
# pytype static type analyzer
.pytype/

# Cython debug symbols
cython_debug/

#учебная строка, добавлена автором
log/
```

— можно сказать, что файл содержит в себе в т. ч. набор относительно простых регулярных выражений. В частности символ «*» означает возможность наличия любых символов. Заключение последовательности символов в квадратные скобки означает возможность присутствия на данном месте любого из них. В частности в строке 3 содержится указание на необходимость игнорирования файлов, имеющих любое имя и одно из следующих расширений: .pyc, .pyo, pyd.

Примерное содержание файла .gitignore, предназначенного для репозитория, содержащего код на языке R:

```
# History files
.Rhistory
.Rapp.history

# Session Data files
.RData

# User-specific files
.Ruserdata

# Example code in package build process
*-Ex.R

# Output files from R CMD build
/*.tar.gz

# Output files from R CMD check
/*.Rcheck/

# RStudio files
.Rproj.user/

# produced vignettes
vignettes/*.html
vignettes/*.pdf

# OAuth2 token, see https://github.com/hadley/httr/releases/
```

```
   tag/v0.3
.httr-oauth

# knitr and R markdown default cache directories
*_cache/
/cache/

# Temporary files created by R markdown
*.utf8.md
*.knit.md

# R Environment Variables
.Renviron

# pkgdown
site docs/

# translation temp files
po/*~

#учебная строка, добавлена автором
log/
```

— используем любой из указанных файлов, сохраним его и проверим статус:

```
$ git status
> On branch master
Changes to be committed:
  (use "git␣restore␣--staged␣<file>..." to unstage)
        new file:   file-1.py

Untracked files:
  (use "git␣add␣<file>..." to include in what will be
     committed)
       .gitignore
```

— как видим папка log пропала и появился файл .gitignore. Добавим его в индекс:

```
$ git add .gitignore
```

— а затем выполним commit:

```
$ git commit -m "Sixth␣commit"
> [master e4adf82] Sixth commit
 2 files changed, 142 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 file-1.py
```

— теперь в случае создания в каталоге любого файла, чьё имя подпадает под правила, описанные в файле .gitignore, он сразу же исключается из списка наблюдения со стороны системы версионирования. Забегая вперёд, можно сказать, что, чаще всего отсутствует необходимость создавать такой файл вручную. Данная функция реализована во многих IDE и будет рассмотрена далее.

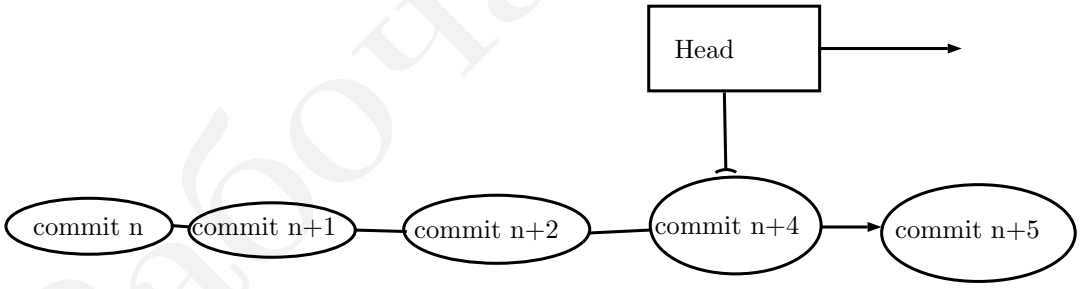
2.3.5. Ветки проекта, указатели branch и Head

В предыдущих подразделах рассматривалась линейная модель созданий версий, которые последовательно формировались одна за другой путём проведений процедуры commit. Git позволяет осуществлять ветвление версий. Посмотрим на текущий статус репозитория:

```
$ git status
> On branch master nothing to commit, working tree clean
```

— обратим внимание на сообщение, возвращённое терминалом, содержащее ссылку на некую branch master. Для того, чтобы разобраться в данном вопросе, следует вспомнить основные принципы работы Git, описанные в подразделах 2.3.1–2.3.2 на с. 31–36. Каждый commit имеет хеш-сумму, содержащую в т.ч. ссылку на предыдущий commit. Таким образом формируется неразрывная цепочка версий. Помимо этого в Git реализована работа указателя Head, представляющего собой метку, указывающую на один из commit. Местонахождение этой метки указывает Git, в каком именно состоянии репозиторий находится в данный момент. При каждом выполнении commit указатель Head смещается на новый commit. Схема работы указателя Head показана на рисунке 2.3.7.

Рис. 2.3.7. Схема работы указателя Head



Установить текущее местонахождение указателя Head можно с помощью команды `git log`.

```
$ git log

> commit e4adf8280c5d95a6f5796dba8e028012565de958
(HEAD -> master)
Author: Kirill Murashev <kirill.murashev@gmail.com>
```

```
Date:   Thu Sep 2 20:42:49 2021 +0300

    Sixth commit

commit 7abee55d2631cf7cf2e94e58f30f36b2be807948 Author:
   Kirill Murashev <kirill.murashev@gmail.com>
Date:   Wed Sep 1 19:52:04 2021 +0300

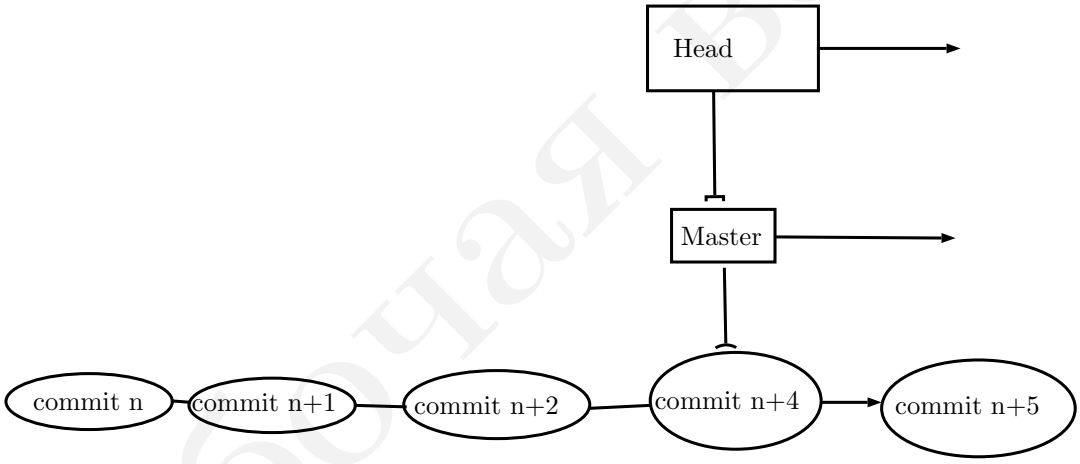
    Fifth commit

commit 284073c521af8b73e16f324698f24040e4b9ee7e

:...skipping...
```

— как следует из ответа терминала, указатель Head находится на последнем шестом commit ветки Master. При этом ветка также представляет собой некий указатель. Таким образом, схема организации указателей выглядит так, как это показано на рисунке 2.3.8.

Рис. 2.3.8. Схема указателей Head и Branch



При наличии достаточной степени развития проекта хорошей практикой считается хранение стабильной версии в ветке Master (в современных системах часто используется наименование Main).[6]

При этом, для новых изменений, находящихся в стадии разработки и тестирования, рекомендуется использовать отдельную ветку. Для создания новой ветки следует использовать команду `git branch <name>`:

```
$ git branch Develop
```

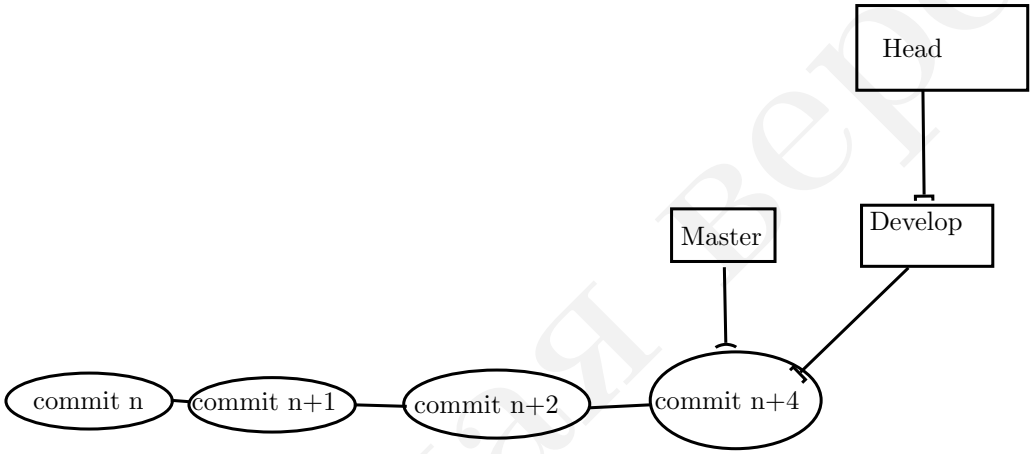
[6] Данное решение обусловлено политическими причинами, поскольку слово Master может ассоциироваться с рабовладением.

— была создана новая ветка Develop. Для перемещения указателья Head на неё используем команду:

```
$ git checkout Develop
```

— состояние репозитория выглядит следующим образом: см. рисунок 2.3.9. Теперь все последующие commit будут сопровождаться указателем ветки Develop, тогда как указатель Master останется на прежнем месте. В случае обратного перемещения указателя Head на ветку Master состояние файлов проекта вернётся к тому, каким оно было в момент создания commit, на который теперь указывает Head. При этом все изменения, сделанные в ветке Develop будут сохранены в ней и доступны в случае перемещения Head на них. После определённого количества перемещений и доработок проект может выглядеть, например так, как показано на рисунке 2.3.10 на следующей странице.

Рис. 2.3.9. Состояние репозитория после переноса указателя Head на ветку Develop



Предположим, что, достигнув состояния, показанного на рисунке на рисунке 2.3.10 на следующей странице, оценщик приходит к выводу о необходимости слияния всех веток в ветку master. Сначала можно посмотреть, какие ветки в принципе существуют.

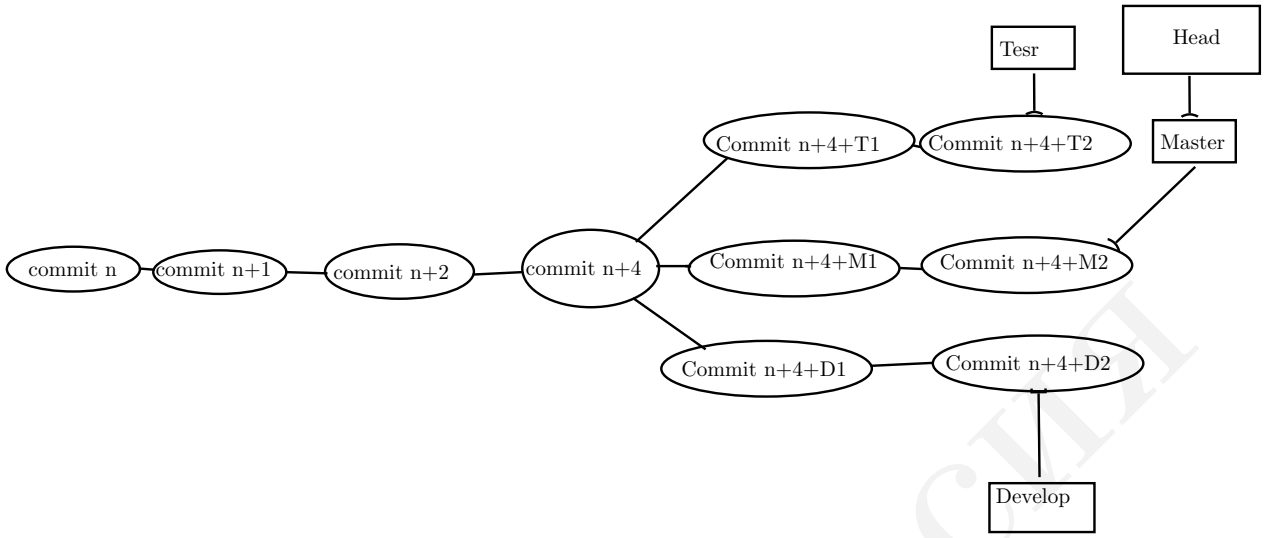
```
$ git branch -a
>
  develop
* master
  test
```

Как видно, указатель Head уже находится на целевой ветке. Если это не так, его следует туда перенести:

```
$ git checkout master
```

После этого выполняем команду `git merge`, в качестве аргумента которой используется имя ветки, которую предполагается объединить с master:

Рис. 2.3.10. Состояние репозитория при наличии нескольких веток




```
$ git merge develop
```

— в случае, когда последний commit из ветки master является прямым родителем для ветки develop объединение происходит путём простого перемещения указателя Head на master, а затем их совместного перемещения на последний commit ветки develop. Данный способ объединения называется fast forward. После такого объединения может быть целесообразным удалить ветку develop, поскольку дальнейшая работа будет вестись в master. Для этого следует выполнить команду:

```
$ git branch -d develop
```

— теперь все изменения, ранее сделанные в ветке develop по-прежнему доступны уже в ветке master. В случае, когда объединяемые ветки не являются родительской и дочерней относительно друг друга процедура объединения происходит более сложным образом. При этом используется та же команда. В результате её выполнения формируется новый commit, называемый merge commit. При этом происходит перемещение указателей master и Head на данный commit. В случае, если commits, являющиеся родительскими по отношению к merge commit имеют только непересекающиеся дополнения относительно последнего общего родительского commit и не имеют взаимоисключающих правок, объединение происходит автоматически и не требует внимания пользователя. Ситуация, при которой имеет место т. н. merge conflict, будет рассмотрена в подразделе 2.3.8 на с. 66–66. Последовательное выполнение команд `git branch` и `git checkout` можно заменить одном командой:

```
$ git chekout -b <branch.name>
```

— в случае необходимости отведения новой ветки не от того commit, на который

указывает Head вторым аргументом этой команды должна быть хеш-сумма того commit, от которого необходимо отвести ветку.

2.3.6. Работа с Github

2.3.6.1. Начало

В материале, изложенном выше в подразделах 2.3.3–2.3.5 2.3.3 на с. 37–62, речь шла о работе с локальным репозиторием, хранящимся на компьютере пользователя. При этом при командной работе часто требуется наличие общего доступа к рабочему каталогу. Также наличие удалённой версии репозитория позволяет распространять разработки на широкую аудиторию. Кроме того, наличие удалённого репозитория позволяет иметь дополнительный бэкап, не зависящий от физического устройства пользователя. Следуя принципу KISS, положенному в основу данной работы, в настоящем разделе будет рассмотрена работа с наиболее популярным сервисом удалённых репозиториев — GitHub [26]. Следует отметить, что существует значительное количество альтернатив, кроме того существует возможность хранения удалённого репозитория на собственном удалённом сервере.

Для начала работы с GitHub необходимо осуществить регистрацию, которая вряд ли может вызвать у кого затруднение в 2021 году. Для создания своего первого репозитория необходимо в меню профиля выбрать пункт Your Repositories и далее создать свой, что также вряд ли может вызвать затруднения. В появившемся меню следует ввести имя репозитория латинскими символами, затем выбрать тип репозитория: публичный либо приватный. В первом случае доступ к репозиторию (но его изменению его содержимого) будет о неограниченного круга пользователей. Для доступа к материалам достаточно иметь ссылку на репозиторий. Во втором случае доступ даже к просмотру будут иметь только те, кому будет предоставлены соответствующие права. Следующие пункты меню позволяют добавить файл Readme, содержащий основные сведения о проекте, файл .gitignore, сформированный по шаблону, разработанному для конкретного языка, а также выбрать лицензию, на условиях которой возможно легальное использование продукта.

Для обеспечения связи между локальным репозиторием и его удалённой версией необходимо зайти в него и выбрать меню Code. В данном меню можно выбрать одно из трёх средств передачи данных:

- протокол HTTPS [78];
- протокол SSH [88];
- средства командной строки GitGub CLI, в свою очередь также реализующие передачу данных посредством протоколов:
  - HTTPS;
  - SSH.

В общем случае рекомендуется использовать протокол SSH. С точки зрения начинающего пользователя различие заключается в том, что при использовании протокола HTTPS каждый раз для соединения с удалённым репозиторием потребуется ввод логина и пароля, тогда как в случае с SSH — нет. При этом для того, чтобы использовать SSH необходимо провести первоначальные настройки. На самом деле, протокол SSH является предпочтительным по ряду технических причин среди которых можно выделить более высокий уровень безопасности, а также эффективное сжатие данных. Для подробного ознакомления с преимуществами и недостатками различных протоколов рекомендуется ознакомиться со следующим официальным материалом [24].

2.3.6.2. Настройка соединения с удалённым репозиторием посредством протокола SSH

Для установления связи с удалённым репозиторием GitHub посредством протокола SSH необходимо осуществить ряд действий, а именно генерировать пару SSH-ключей, а затем добавить их в профиль аккаунта на портале GitHub.

2.3.6.2.1. Проверка наличия существующих SSH-ключей. Для начала необходимо проверить наличие существующих ключей. Для этого следует запустить Терминал и ввести команду:

```
$ ls -al ~/.ssh
```

— в случае наличия существующих ключей Терминал возвратит примерно следующее сообщение:

```
> total 20
drwx------  2 user.name user.name 4096 aug   14 14:42 .
drwxr-xr-x 36 user.name user.name 4096 sept   1 09:14 ..
-rw-------  1 user.name user.name  464 aug   11 11:05
   id_ed25519
-rw-r--r--  1 user.name user.name  107 aug   11 10:04
   id_ed25519.pub
-rw-r--r--  1 user.name user.name 1326 aug   11 19:08
   known_hosts
```

— в этом случае можно пропустить второй этап, описанный в подсекции 2.3.6.2.2–65, и перейти к третьему этапу, описанному в подсекции 2.3.6.2.3 на с. 65–66. В случае отсутствия существующей пары необходимо осуществить её генерацию.

2.3.6.2.2. Генерация новой пары ключей. Для создания пары ключей на основе алгоритма RSA, необходимо запустить Терминал и выполнить команду:

```
$ ssh-keygen -t rsa -b 4096 -C "user.name@host.com"
```

— указав при этот тот адрес электронной почты, который указан в профиле на GitHub. Помимо адреса электронной почты аргументами команды являются: алгоритм генерации ключа и его длина в битах. Следует сказать, что алгоритм RSA не является единственным О различиях между алгоритмами RSA [86], DSA [73], ECDSA [74] и Ed25519 [11] можно почитать в следующих статьях и комментариях к ним: [19, 63, 58]. В целом, можно сказать, что для целей обучения анализу данных, равно как и для большинства практических целей оценщиков нет существенной разницы в том, какой алгоритм будет использован при создании пары ключей. Однако, с точки зрения соответствия лучшим практикам и современным тенденциям можно сказать следующее:

a) Алгоритм DSA несколько устарел и подвержен уязвимости, поскольку решение проблемы вычислительной сложности взятия логарифмов в конечных полях [94], на которой он и был основан, было найдено в 2013 году.

b) Схожий с DSA алгоритм ECDSA лишён указанного недостатка, поскольку основан не на конечном числовом поле [97], а на группе точек эллиптической кривой [110]. При этом криптографическая стойкость алгоритма в существенной степени зависит от возможности компьютера генерировать случайные числа.

c) Алгоритм RSA обеспечивает достаточную надёжность при условии достаточной длины ключа. Длина ключа в 3072 либо 4096 бит является достаточной. Данный алгоритм является рекомендуемым в том случае, если нет возможности использовать алгоритм Ed25519.

d) Алгоритм Ed25519 является предпочтительным во всех случаях, когда система технически способна работать с ним. Данный алгоритм обеспечивает хорошую криптостойкость, при этом работа с ключами происходит существенно быстрее, чем при использовании алгоритма RSA. Длина публичного ключа составляет всего 68 символов, тогда как RSA генерирует публичный ключ длиной в 544 символа (при 3072 бит).

Таким образом, вместо вышеуказанной команды рекомендуется использовать команду:

```
$ ssh-keygen -t ed25519 -C "user.name@host.com"
```

— адрес электронной почты также должен совпадать с тем, который указан в профиле на портале GitHub. Терминал возвратит сообщение:

```
> Enter a file in which to save the key (/home/user.name/.
  ssh/id_ed25519): [Press enter]
```

— предложив нажать Enter для сохранения ключей в каталоге по умолчанию. Следует согласиться с предложением и перейти к этапу создания пароля:

```
> Enter passphrase (empty for no passphrase): [Type a
   passphrase]
> Enter same passphrase again: [Type passphrase again]
```

— ключи SSH готовы. Для возможности работы с ними необходимо добавить их в ssh-agent. Для этого сначала необходимо запустить ssh-agent в фоновом режиме, выполнив последовательно две команды:

```
$ sudo -s -H
$ eval "$(ssh-agent␣-s)"
```

— далее осуществляется добавление самого ключа:

```
$ ssh-add ~/.ssh/id_ed25519
```

— ключи зарегистрированы в ssh-agent и могут быть использованы для взаимодействия с порталом GitHub.

2.3.6.2.3. Добавление публичного ключа на портал GitHub. Для того, чтобы добавить в профиль на портале GitHub публичный ssh ключ необходимо получить его значение. Для начала следует установить xclip:

```
$ sudo apt-get update
$ sudo apt-get install xclip
```

— теперь существует возможность автоматически копировать возвращаемые терминалом сообщения в буфер обмена. Сделаем это:

```
xclip -selection clipboard < ~/.ssh/id_ed25519.pub
```

— в данный момент буфер обмена содержит значение публичного ssh ключа. После этого необходимо зайти в свой профиль на портале GitHub и найти пункт меню **Settings**, а затем **SSH and GPG keys**. После этого следует нажать на кнопку **New SSH key**. Откроется меню, состоящее из двух полей: заголовка и значение ключа. В поле заголовка можно ввести любые символы, например имя, фамилию и должность. В поле значения ключа необходимо вставить содержимое буфера обмена. Существует семь возможных начальных символов ключа:

- «ssh-rsa»;
- «ecdsa-sha2-nistp256»;
- «ecdsa-sha2-nistp384»;
- «ecdsa-sha2-nistp521»;
- «ssh-ed25519»;
- «sk-ecdsa-sha2-nistp256@openssh.com»;

- «sk-«ssh-ed25519@openssh.com»

— в зависимости от применённого алгоритма. В случае совпадения практического значения с одним из возможных, можно сделать вывод о том, что все подготовительные операции были выполнены корректно. Нет необходимости вглядываться в имеющееся на практике значение: система в любом случае не зарегистрирует ключ, не отвечающий требованиям по маске. В том случае, если ключ прошёл валидацию, кнопка Add SHH key, расположенная ниже поля, станет активной. После её нажатия произойдёт добавление ключа.

Перед началом использования связи по SSH протоколу рекомендуется провести проверку. Для этого в терминале следует ввести команду:

```
$ ssh -T git@github.com
```

— терминал запросит ввести пароль, установленный при генерации ключей. В случае установления успешной связи терминал возвратит сообщение:

```
Hi Kirill-Murashev! You've successfully authenticated, but
   GitHub does not provide shell access.
```

— связь установлена, возможна работа с удалённым репозиторием.

2.3.6.2.4. Создание и установка GPG ключа. End

2.3.7. Rebase

2.3.8. Работа с Git в IDE

End
End

2.4. Установка и настройка

2.4.1. Git

2.4.1.1. Установка на операционных системах, основанных на Debian: Debian, Ubuntu, Mint и т. п.

В операционных системах, основанных на ядре Linux [56], относящихся к ветке Debian [81], Git зачастую бывает уже установлен вместе с системой. Чтобы проверить наличие Git в командную строку терминала следует ввести:

```
git
```

В случае наличия Git в системе, терминал возвратит длинное сообщение, начинающееся примерно следующим образом:

```
usage: git [--version] [--help] [-C <path>] [-c <name>=<
   value>]
[--exec-path[=<path>]] [--html-path] [--man-path] [--info-
   path]            [-p | --paginate | -P | --no-pager] [--
   no-replace-objects] [--bare]            [--git-dir=<path
   >] [--work-tree=<path>] [--namespace=<name>]
```

В случае его отсутствия:

```
Command 'git' not found, did you mean:
```

Во втором случае следует использовать следующие команды:

```
sudo apt update -y
sudo apt install git -y
```

Процесс проходит автоматически и не требует внимания со стороны пользователя.

2.4.1.2. Установка на операционной системе Windows

Установка Git на Windows осуществляется обычным для данной операционной системы образом. Необходимо загрузить установочный файл с соответствующей страницы [21] и запустить процесс установки, желательно приняв при этом все настройки по умолчанию.

2.4.1.3. Установка на macOS

Существует несколько способов установки Git на macOS. Их перечень приведён на соответствующей странице [22] сайта Git. Следует отметить, что в случае наличия в системе Xcode [111] Git также уже присутствует, и его установка не требуется. В данном материала приводится один из возможных способов. Для начала необходимо установить менеджер пакетов Homebrew [29]. Для этого в командной строке терминала необходимо ввести следующую команду:

```
/bin/bash -c "$(curl␣-fsSL␣https://raw.githubusercontent.com
   /Homebrew/install/HEAD/install.sh)"
```

После этого можно перейти к установке самого Git. Для этого в командной строке терминала необходимо ввести следующую команду:

```
brew install git
```

Как и в случае, описанном выше в секции 2.4.1.1 на предшествующей странице–67, процесс проходит автоматически и не требует внимания со стороны пользователя.

R

RStudio

Python

End

The End