Contents

Change history ... xviii
- 1. (1.0.0) February 25th 2019 ... xviii
- 2. (1.0.1) May 22nd 2019 ... xviii
- 3. (1.0.2) October 31st 2019 ... xviii
- 4. (1.0.3) April 10th 2020 ... xix
- 5. (1.0.4) June 18th 2021 ... xix

License and acknowledgements ... xx

1. Introduction ... 1
- 1.1. Prerequisites ... 2
- 1.2. How the book is structured ... 2
- 1.3. Typographic conventions ... 3
- 1.4. The Jit codebase ... 4

2. Getting to know `.git` ... 5
- 2.1. The `.git` directory ... 5
  - 2.1.1. `.git/config` ... 6
  - 2.1.2. `.git/description` ... 7
  - 2.1.3. `.git/HEAD` ... 7
  - 2.1.4. `.git/info` ... 7
  - 2.1.5. `.git/hooks` ... 8
  - 2.1.6. `.git/objects` ... 8
  - 2.1.7. `.git/refs` ... 9
- 2.2. A simple commit ... 9
  - 2.2.1. `.git/COMMIT_EDITMSG` ... 11
  - 2.2.2. `.git/index` ... 11
  - 2.2.3. `.git/logs` ... 12
  - 2.2.4. `.git/refs/heads/master` ... 12
- 2.3. Storing objects ... 12
  - 2.3.1. The `cat-file` command ... 13
  - 2.3.2. Blobs on disk ... 14
  - 2.3.3. Trees on disk ... 16
  - 2.3.4. Commits on disk ... 18
  - 2.3.5. Computing object IDs ... 19
  - 2.3.6. Problems with SHA-1 ... 21
- 2.4. The bare essentials ... 24

I. Storing changes ... 25
- 3. The first commit ... 26
  - 3.1. Initialising a repository ... 26
    - 3.1.1. A basic `init` implementation ... 27
    - 3.1.2. Handling errors ... 28
    - 3.1.3. Running Jit for the first time ... 29
  - 3.2. The `commit` command ... 30
    - 3.2.1. Storing blobs ... 31
    - 3.2.2. Storing trees ... 36
    - 3.2.3. Storing commits ... 39
- 4. Making history ... 45
  - 4.1. The `parent` field ... 45

4.1.1. A link to the past ... 46
4.1.2. Differences between trees ... 46
4.2. Implementing the `parent` chain ... 47
4.2.1. Safely updating `.git/HEAD` ... 49
4.2.2. Concurrency and the filesystem ... 52
4.3. Don't overwrite objects ... 53
5. Growing trees ... 55
5.1. Executable files ... 55
5.1.1. File modes ... 55
5.1.2. Storing executables in trees ... 58
5.2. Nested trees ... 60
5.2.1. Recursive trees in Git ... 61
5.2.2. Building a Merkle tree ... 63
5.2.3. Flat or nested? ... 70
5.3. Reorganising the project ... 71
6. The index ... 73
6.1. The `add` command ... 73
6.2. Inspecting `.git/index` ... 74
6.3. Basic `add` implementation ... 77
6.4. Storing multiple entries ... 82
6.5. Adding files from directories ... 84
7. Incremental change ... 87
7.1. Modifying the index ... 87
7.1.1. Parsing `.git/index` ... 87
7.1.2. Storing updates ... 92
7.2. Committing from the index ... 93
7.3. Stop making sense ... 96
7.3.1. Starting a test suite ... 96
7.3.2. Replacing a file with a directory ... 98
7.3.3. Replacing a directory with a file ... 100
7.4. Handling bad inputs ... 104
7.4.1. Non-existent files ... 105
7.4.2. Unreadable files ... 107
7.4.3. Locked index file ... 108
8. First-class commands ... 111
8.1. Abstracting the repository ... 112
8.2. Commands as classes ... 114
8.2.1. Injecting dependencies ... 116
8.3. Testing the commands ... 122
8.4. Refactoring the commands ... 127
8.4.1. Extracting common code ... 127
8.4.2. Reorganising the `add` command ... 127
9. Status report ... 131
9.1. Untracked files ... 131
9.1.1. Untracked files not in the index ... 133
9.1.2. Untracked directories ... 135
9.1.3. Empty untracked directories ... 139
9.2. Index/workspace differences ... 141

9.2.1. Changed contents ... 142
9.2.2. Changed mode ... 144
9.2.3. Size-preserving changes ... 146
9.2.4. Timestamp optimisation ... 149
9.2.5. Deleted files ... 151
10. The next commit ... 156
10.1. Reading from the database ... 156
10.1.1. Parsing blobs ... 158
10.1.2. Parsing commits ... 158
10.1.3. Parsing trees ... 159
10.1.4. Listing the files in a commit ... 162
10.2. `HEAD`/index differences ... 162
10.2.1. Added files ... 163
10.2.2. Modified files ... 166
10.2.3. Deleted files ... 167
10.3. The long format ... 169
10.3.1. Making the change easy ... 171
10.3.2. Making the easy change ... 173
10.3.3. Orderly change ... 176
10.4. Printing in colour ... 177
11. The Myers diff algorithm ... 182
11.1. What's in a diff? ... 182
11.2. Time for some graph theory ... 184
11.2.1. Walking the graph ... 186
11.2.2. A change of perspective ... 190
11.2.3. Implementing the shortest-edit search ... 193
11.3. Retracing our steps ... 195
11.3.1. Recording the search ... 198
11.3.2. And you may ask yourself, how did I get here? ... 199
12. Spot the difference ... 202
12.1. Reusing `status` ... 202
12.2. Just the headlines ... 205
12.2.1. Unstaged changes ... 206
12.2.2. A common pattern ... 210
12.2.3. Staged changes ... 212
12.3. Displaying edits ... 215
12.3.1. Splitting edits into hunks ... 218
12.3.2. Displaying diffs in colour ... 226
12.3.3. Invoking the pager ... 228
II. Branching and merging ... 233
13. Branching out ... 234
13.1. Examining the `branch` command ... 236
13.2. Creating a branch ... 239
13.3. Setting the start point ... 242
13.3.1. Parsing revisions ... 243
13.3.2. Interpreting the AST ... 245
13.3.3. Revisions and object IDs ... 248
14. Migrating between trees ... 255

14.1. Telling trees apart 256
14.2. Planning the changes 261
14.3. Updating the workspace 264
14.4. Updating the index 266
14.5. Preventing conflicts 267
14.5.1. Single-file status checks 268
14.5.2. Checking the migration for conflicts 269
14.5.3. Reporting conflicts 272
14.6. The perils of self-hosting 274
15. Switching branches 276
15.1. Symbolic references 278
15.1.1. Tracking branch pointers 279
15.1.2. Detached `HEAD` 280
15.1.3. Retaining detached histories 282
15.2. Linking `HEAD` on checkout 283
15.2.1. Reading symbolic references 284
15.3. Printing checkout results 285
15.4. Updating `HEAD` on commit 289
15.4.1. The `master` branch 291
15.5. Branch management 292
15.5.1. Parsing command-line options 293
15.5.2. Listing branches 295
15.5.3. Deleting branches 299
16. Reviewing history 302
16.1. Linear history 302
16.1.1. Medium format 304
16.1.2. Abbreviated commit IDs 305
16.1.3. One-line format 306
16.1.4. Branch decoration 307
16.1.5. Displaying patches 310
16.2. Branching histories 314
16.2.1. Revision lists 315
16.2.2. Logging multiple branches 316
16.2.3. Excluding branches 322
16.2.4. Filtering by changed paths 329
17. Basic merging 335
17.1. What is a merge? 335
17.1.1. Merging single commits 335
17.1.2. Merging a chain of commits 337
17.1.3. Interpreting merges 339
17.2. Finding the best common ancestor 342
17.3. Commits with multiple parents 344
17.4. Performing a merge 346
17.5. Best common ancestors with merges 348
17.6. Logs in a merging history 352
17.6.1. Following all commit parents 352
17.6.2. Hiding patches for merge commits 353
17.6.3. Pruning treesame commits 353

17.6.4. Following only treesame parents ... 354
17.7. Revisions with multiple parents ... 354
18. When merges fail ... 356
18.1. A little refactoring ... 356
18.2. Null and fast-forward merges ... 358
18.2.1. Merging an existing ancestor ... 358
18.2.2. Fast-forward merge ... 359
18.3. Conflicted index entries ... 361
18.3.1. Inspecting the conflicted repository ... 362
18.3.2. Stages in the index ... 363
18.3.3. Storing entries by stage ... 365
18.3.4. Storing conflicts ... 366
18.4. Conflict detection ... 368
18.4.1. Concurrency, causality and locks ... 369
18.4.2. Add/edit/delete conflicts ... 371
18.4.3. File/directory conflicts ... 375
19. Conflict resolution ... 378
19.1. Printing conflict warnings ... 378
19.2. Conflicted status ... 381
19.2.1. Long status format ... 382
19.2.2. Porcelain status format ... 383
19.3. Conflicted diffs ... 384
19.3.1. Unmerged paths ... 384
19.3.2. Selecting stages ... 385
19.4. Resuming a merge ... 387
19.4.1. Resolving conflicts in the index ... 387
19.4.2. Retaining state across commands ... 387
19.4.3. Writing a merge commit ... 389
20. Merging inside files ... 392
20.1. The diff3 algorithm ... 394
20.1.1. Worked example ... 394
20.1.2. Implementing diff3 ... 396
20.1.3. Using diff3 during a merge ... 402
20.2. Logging merge commits ... 403
20.2.1. Unifying hunks ... 409
20.2.2. Diffs during merge conflicts ... 412
20.2.3. Diffs for merge commits ... 413
21. Correcting mistakes ... 415
21.1. Removing files from the index ... 415
21.1.1. Preventing data loss ... 416
21.1.2. Refinements to the `rm` command ... 419
21.2. Resetting the index state ... 422
21.2.1. Resetting to a different commit ... 425
21.3. Discarding commits from your branch ... 426
21.3.1. Hard reset ... 428
21.3.2. I'm losing my `HEAD` ... 432
21.4. Escaping from merges ... 433
22. Editing messages ... 436

22.1. Setting the commit message 437
22.2. Composing the commit message 438
22.2.1. Launching the editor 440
22.2.2. Starting and resuming merges 442
22.3. Reusing messages 445
22.3.1. Amending the `HEAD` 448
22.3.2. Recording the committer 450
23. Cherry-picking 454
23.1. Cherry-picking a single commit 457
23.1.1. New types of pending commit 459
23.1.2. Resuming from conflicts 461
23.2. Multiple commits and ranges 464
23.2.1. Rev-list without walking 464
23.2.2. Conflicts during ranges 466
23.2.3. When all else fails 471
24. Reshaping history 475
24.1. Changing old commits 475
24.1.1. Amending an old commit 475
24.1.2. Reordering commits 476
24.2. Rebase 478
24.2.1. Rebase onto a different branch 480
24.2.2. Interactive rebase 481
24.3. Reverting existing commits 485
24.3.1. Cherry-pick in reverse 487
24.3.2. Sequencing infrastructure 488
24.3.3. The `revert` command 490
24.3.4. Pending commit status 494
24.3.5. Reverting merge commits 495
24.4. Stashing changes 497
III. Distribution 501
25. Configuration 502
25.1. The Git config format 502
25.1.1. Whitespace and comments 503
25.1.2. Abstract and concrete representation 505
25.2. Modelling the `.git/config` file 507
25.2.1. Parsing the configuration 508
25.2.2. Manipulating the settings 511
25.2.3. The configuration stack 515
25.3. Applications 517
25.3.1. Launching the editor 517
25.3.2. Setting user details 518
25.3.3. Changing diff formatting 518
25.3.4. Cherry-picking merge commits 520
26. Remote repositories 524
26.1. Storing remote references 525
26.2. The `remote` command 526
26.2.1. Adding a remote 527
26.2.2. Removing a remote 529

26.2.3. Listing remotes 529
26.3. Refspecs 531
26.4. Finding objects 534
27. The network protocol 540
27.1. Programs as ad-hoc servers 540
27.2. Remote agents 542
27.3. The packet-line protocol 543
27.4. The pack format 547
27.4.1. Writing packs 548
27.4.2. Reading from packs 552
27.4.3. Reading from a stream 556
28. Fetching content 561
28.1. Pack negotiation 561
28.1.1. Non-fast-forward updates 563
28.2. The `fetch` and `upload-pack` commands 565
28.2.1. Connecting to the remote 567
28.2.2. Transferring references 568
28.2.3. Negotiating the pack 569
28.2.4. Sending object packs 572
28.2.5. Updating remote refs 573
28.2.6. Connecting to remote repositories 577
28.3. Clone and pull 578
28.3.1. Pulling and rebasing 581
28.3.2. Historic disagreement 583
29. Pushing changes 585
29.1. Shorthand refspecs 585
29.2. The `push` and `receive-pack` commands 587
29.2.1. Sending update requests 589
29.2.2. Updating remote refs 594
29.2.3. Validating update requests 599
29.3. Progress meters 601
30. Delta compression 608
30.1. The XDelta algorithm 608
30.1.1. Comparison with diffs 611
30.1.2. Implementation 612
30.2. Delta encoding 616
30.3. Expanding deltas 620
31. Compressing packs 623
31.1. Finding similar objects 623
31.1.1. Generating object paths 626
31.1.2. Sorting packed objects 627
31.2. Forming delta pairs 630
31.2.1. Sliding-window compression 632
31.2.2. Limiting delta chain length 635
31.3. Writing and reading deltas 638
32. Packs in the database 644
32.1. Indexing packs 645
32.1.1. Extracting `TempFile` 646

32.1.2. Processing the incoming pack 648
32.1.3. Generating the index 649
32.1.4. Reconstructing objects 652
32.1.5. Storing the index 653
32.2. A new database backend 657
32.2.1. Reading the pack index 659
32.2.2. Replacing the backend 662
32.3. Offset deltas 668
33. Working with remote branches 669
33.1. Remote-tracking branches 669
33.1.1. Logging remote branches 669
33.1.2. Listing remote branches 671
33.2. Upstream branches 672
33.2.1. Setting an upstream branch 673
33.2.2. Safely deleting branches 678
33.2.3. Upstream branch divergence 679
33.2.4. The `@{upstream}` revision 682
33.2.5. Fetching and pushing upstream 683
34. …and everything else 685
IV. Appendices 687
A. Programming in Ruby 688
A.1. Installation 688
A.2. Core language 689
A.2.1. Control flow 689
A.2.2. Error handling 692
A.2.3. Objects, classes, and methods 693
A.2.4. Blocks 699
A.2.5. Constants 700
A.3. Built-in data types 701
A.3.1. `true`, `false` and `nil` 701
A.3.2. `Integer` 701
A.3.3. `String` 702
A.3.4. `Regexp` 703
A.3.5. `Symbol` 703
A.3.6. `Array` 703
A.3.7. `Range` 705
A.3.8. `Hash` 706
A.3.9. `Struct` 707
A.4. Mixins 707
A.4.1. `Enumerable` 708
A.4.2. `Comparable` 710
A.5. Libraries 711
A.5.1. `Digest` 711
A.5.2. `FileUtils` 711
A.5.3. `Forwardable` 712
A.5.4. `Open3` 712
A.5.5. `OptionParser` 712
A.5.6. `Pathname` 712

A.5.7. `Set` 713
A.5.8. `Shellwords` 713
A.5.9. `StringIO` 713
A.5.10. `StringScanner` 714
A.5.11. `Time` 714
A.5.12. `URI` 714
A.5.13. `Zlib` 714
B. Bitwise arithmetic 715

List of Figures

2.1. Contents of `.git` after running `git init` ... 6
2.2. Contents of `.git` after a single commit ... 10
2.3. Files generated by the first commit ... 11
2.4. Creating a shell alias ... 15
2.5. Decimal, hexadecimal and binary representations of numbers ... 18
2.6. Minimum viable Git repository ... 24
5.1. Interpreting the bits of a file mode ... 56
5.2. Octal digit file permissions ... 57
5.3. Jit project file layout ... 60
5.4. Project layout with `jit` in a directory ... 61
5.5. Tree containing a flat list of entries ... 65
5.6. Tree containing a nested tree ... 65
5.7. Project organised into directories ... 72
11.1. Three equivalent diffs ... 183
11.2. Interleaving deletions and insertions ... 184
11.3. Aligning diff changes with logical code blocks ... 184
11.4. Edit graph for converting `ABCABBA` into `CBABAC` ... 185
11.5. Initial graph exploration state ... 186
11.6. Exploration state after a single move ... 186
11.7. Partial exploration state for two moves ... 186
11.8. Complete exploration state after two moves ... 187
11.9. First explorations for the third move ... 187
11.10. Second explorations for the third move ... 187
11.11. Discarding the worse result ... 188
11.12. Complete exploration state after three moves ... 188
11.13. First explorations for the fourth move ... 188
11.14. Second explorations for the fourth move ... 189
11.15. Complete exploration state after four moves ... 189
11.16. Partial exploration state for five moves ... 189
11.17. Reaching the bottom-right of the graph ... 190
11.18. Exploration state in *d*-*k* space ... 190
11.19. Picking a move from the higher *x* value ... 191
11.20. Picking a rightward move when both predecessors have equal *x* ... 191
11.21. Exploration state with redundant data removed ... 192
11.22. State array after each move ... 192
11.23. Completed exploration state for `diff(ABCABBA, CBABAC)` ... 195
11.24. Backtracking choices for $(d, k) = (5,1)$ ... 196
11.25. Choosing the previous highest *x* value ... 196
11.26. Backtracking from $(d, k) = (4,2)$ ... 197
11.27. Backtracking from $(d, k) = (3,1)$ ... 197
11.28. Backtracking to the starting position (0,0) ... 198
12.1. List of edits in a diff ... 221
12.2. Scanning to find the first change ... 222
12.3. Setting up to begin a hunk ... 222
12.4. Capturing the first context line ... 223
12.5. Expanding the hunk on reading a change ... 224

12.6. Detecting the end of a group of changes 224
13.1. Commits forming a linked list 234
13.2. Moving `HEAD` to an older commit 234
13.3. Commits diverging from the original history 234
13.4. `master` branch referring to the original history 234
13.5. Named branches with diverging histories 235
13.6. Two copies of the same history 235
13.7. Chains of commits built on a common history 235
13.8. Branches from a shared history 236
13.9. Alice and Bob merging each other's changes 236
13.10. `.git/logs` and `.git/refs` for a single `master` branch 237
13.11. `.git/logs` and `.git/refs` after creating the `topic` branch 237
14.1. Chain of commits 255
14.2. New branch pointing to the last commit 255
14.3. New branch pointing to an older commit 255
14.4. Moving `HEAD` to a different branch 255
14.5. Trees of two adjacent commits 257
14.6. Top level of trees from two commits 257
14.7. Trees for the `lib` sub-tree of two commits 257
14.8. Trees for the `lib/models` sub-tree of two commits 258
15.1. Two branch pointers into a commit history 276
15.2. Moving `HEAD` to point at an older commit 276
15.3. Committing moves the `HEAD` pointer 276
15.4. Committing a chain of commits 277
15.5. Moving `HEAD` to another branch 277
15.6. Branch pointer following a commit chain 277
15.7. Two branch pointers into a commit history 278
15.8. `HEAD` referring to a branch pointer 279
15.9. Branch tracking newly created commits 280
15.10. Switching `HEAD` to point at a different branch 280
15.11. Detached `HEAD`, pointing directly to a commit 281
15.12. Branch formed while in detached `HEAD` mode 282
15.13. Reattaching `HEAD` to a new branch pointer 283
15.14. `HEAD` attached to a branch 289
15.15. Detaching `HEAD` by making a commit 289
15.16. Branch pointer following a new commit 290
16.1. Single chain of commits 303
16.2. History with two branches 316
16.3. Starting commits for the history search 317
16.4. Reading the start commits to find their parents 317
16.5. Continuing along the `topic` branch 317
16.6. Continuing along the `master` branch 318
16.7. Locating the common ancestor 318
16.8. Starting commits for a new `log` search 318
16.9. Processing the most recent commit 319
16.10. Processing the second youngest commit 319
16.11. Processing the third youngest commit 319
16.12. History with two branches 322

16.13. History reachable from `master` ... 322
16.14. History reachable from `topic` ... 323
16.15. History reachable from `master` and not from `topic` ... 323
16.16. History containing merge commits ... 323
16.17. Initial state for a range search ... 324
16.18. Processing an excluded commit ... 324
16.19. Processing an included commit ... 324
16.20. Processing an uninteresting commit ... 324
16.21. Stopping the search on an all-uninteresting queue ... 325
16.22. Commit graph with a long branch ... 327
16.23. Loaded commits part-way through a graph search ... 328
17.1. Two branches changing different files ... 336
17.2. History following a successful merge ... 337
17.3. Two branches changing multiple files ... 338
17.4. History after merging a chain of commits ... 339
17.5. Alice and Bob's trees before the merge ... 340
17.6. Calculating Alice and Bob's changes since they diverged ... 340
17.7. Symmetric merge result ... 341
17.8. History leading to a merge commit ... 341
17.9. Merged history with further commits ... 342
17.10. Merged history without merged parent links ... 342
17.11. History following multiple merges ... 342
17.12. Branch containing fork/merge bubbles ... 347
17.13. Branching/merging history ... 348
17.14. History with many candidate common ancestors ... 349
18.1. Merging branches changing different files ... 356
18.2. Merging branches changing the same file ... 356
18.3. History with `topic` already merged into `master` ... 358
18.4. History where `HEAD` is an ancestor of another branch ... 359
18.5. Fast-forwarding the `HEAD` branch ... 360
18.6. Flattened fast-forward result state ... 360
18.7. Forced merge commit created using `--no-ff` ... 360
18.8. Merging branches changing the same file ... 362
18.9. Bitwise breakdown of the index entry flag bytes ... 365
18.10. Merging branches changing different files ... 369
18.11. Sequential application of changes from parallel branches ... 369
18.12. Reversed sequential application of changes from parallel branches ... 369
18.13. Merging branches changing the same file ... 370
18.14. Sequential application of changes to the same file ... 370
18.15. Reversed sequential application of changes to the same file ... 370
20.1. Merging branches changing the same file ... 392
20.2. Initial file contents ... 393
20.3. Inserting a new line and deleting another ... 393
20.4. Deleting a line and then inserting one ... 393
20.5. Original ingredient list ... 394
20.6. Alice and Bob's changes to the ingredient list ... 394
20.7. Alice's diff against the original ... 395
20.8. Bob's diff against the original ... 395

20.9. Aligning the three versions on unchanged lines 395
21.1. Chain of commits with `HEAD` pointing to the branch tip 426
21.2. Checking out an earlier commit, leaving the branch pointer in place 426
21.3. Deleting the old branch pointer and restarting it at the current `HEAD` 427
21.4. Checking out the moved branch 427
21.5. Chain of unreachable commits 432
21.6. `ORIG_HEAD` pointing to the previous `HEAD` position 432
22.1. Commit chain before a reset 447
22.2. Commit chain after a reset 447
22.3. Commit graph with an 'amended' commit 447
22.4. Chain of commits to be squashed 448
22.5. Desired squashed history 448
22.6. `HEAD` reset to before the commits to be squashed 448
22.7. Squashed history 448
22.8. The `commit` command creates a new commit 448
22.9. The `commit --amend` command replaces an existing commit 449
23.1. History with two branches 454
23.2. Merging part of the `topic` branch into `master` 454
23.3. Cherry-picking a single commit into `master` 454
23.4. Imagined history graph for a cherry-pick merge 455
23.5. Merge input commits with history links removed 455
23.6. Merge commit pointing to the two input commits 456
23.7. Merge commit following a previous merge 456
23.8. Cherry-pick commit unconnected to its source 456
23.9. History with conflicting changes 466
23.10. Partially completed range cherry-pick 466
23.11. Partially completed range cherry-pick with conflicts resolved 467
24.1. Sequence of five commits 475
24.2. Resetting `HEAD` to the target commit 475
24.3. Amending the target commit 476
24.4. Cherry-picking the remaining history 476
24.5. Sequence of six commits 476
24.6. Creating a fixup branch 477
24.7. Cherry-picking the reordered commits 477
24.8. Cherry-picking the remaining history 478
24.9. Resetting the original branch 478
24.10. History with two divergent branches 478
24.11. Branch after rebasing to the tip of `master` 479
24.12. Resetting the current branch to the upstream branch 479
24.13. Cherry-picking the branch onto the upstream 480
24.14. History with three chained branches 480
24.15. Rebase onto a different branch 480
24.16. Resetting to the target branch 481
24.17. Cherry-picking the original branch 481
24.18. History before squashing 482
24.19. Checking out the desired tree 482
24.20. Resetting to the parent commit 482
24.21. Amending the parent commit to contain the squashed changes 483

24.22. Cherry-picking the remaining history 483
24.23. Cleaning up branch pointers after squashing 483
24.24. History before a fix-up 484
24.25. Creating a fix-up commit 484
24.26. Starting a fix-up branch 484
24.27. Cherry-picking the fix-up commit 484
24.28. Creating a squashed commit containing the fix-up 485
24.29. History following a relocated fix-up commit 485
24.30. History with concurrent edits 486
24.31. Alice removes a shared commit from the history 486
24.32. Merging reintroduces a dropped commit 486
24.33. Committing to undo earlier changes 487
24.34. Merging does not reintroduce the removed content 487
24.35. History with two files 487
24.36. Reverting an old change 488
24.37. History with non-commutative commits 491
24.38. Reverting the last two commits 491
24.39. Branched and merged history 496
24.40. History with reverted merge 496
24.41. Attempting to re-merge a reverted branch 497
24.42. Cherry-picking reverted changes 497
24.43. Reverting a reverted merge 497
24.44. Stored pair of stash commits 498
24.45. Initial work state 498
24.46. Checking out the `stash` branch 498
24.47. Committing the index state 498
24.48. Committing the workspace state 499
24.49. Checking out the original branch 499
24.50. Adding more commits to `master` 499
24.51. Cherry-picking the stash commits 500
24.52. Regenerating the uncommitted changes 500
26.1. Alice’s history 532
26.2. Bob’s repository after fetching from Alice 532
26.3. Alice’s extended history 532
26.4. Bob’s updated repository 532
26.5. Alice’s amended history 532
26.6. Bob’s repository after an unforced fetch 532
26.7. Bob’s repository after a forced fetch 533
26.8. Forking history 535
28.1. Alice’s initial repository 561
28.2. Bob’s copy of Alice’s repository 561
28.3. Alice extends the `master` branch 561
28.4. Bob’s updated database 562
28.5. Bob’s updates his remote reference 562
28.6. Alice’s rewritten history 563
28.7. Alice’s database with unreachable commits removed 564
28.8. Bob’s repository after fetching from Alice 579
28.9. Bob fetches new commits from Alice 579

28.10. Bob performs a fast-forward merge ... 579
28.11. Bob fetches divergent history ... 580
28.12. Bob merges Alice's changes into his `master` branch ... 580
28.13. Alice's history ... 580
28.14. Alice fetches from Bob ... 580
28.15. Alice fast-forwards to Bob's merge commit ... 581
28.16. Alice merges Bob's changes with her own ... 581
28.17. Bob's repository after fetching from Alice ... 582
28.18. Bob resets his `master` branch to Alice's latest commit ... 582
28.19. Bob rebases his branch ... 582
28.20. Alice's repository after fetching Bob's rebased commits ... 582
28.21. Alice fast-forwards her `master` branch ... 583
28.22. Bob's repository after fetching from Alice ... 583
28.23. Bob fetches Alice's rewritten history ... 583
28.24. Bob merges Alice's old and new histories ... 583
28.25. Bob's repository after fetching Alice's revert commits ... 584
29.1. Repository in sync with the remote ... 591
29.2. Local history diverges from the remote ... 592
29.3. Merging the remote's changes ... 592
29.4. Alice and Bob's combined history ... 595
29.5. Alice's update is accepted ... 595
29.6. Bob's repository after merging Alice's changes ... 595
30.1. First few iterations of the matching loop ... 610
30.2. Finding the first matching block ... 610
31.1. Empty sliding window ... 630
31.2. Sliding window with one element ... 630
31.3. Sliding window with two elements ... 631
31.4. Sliding window with three elements ... 631
31.5. Sliding window with one slot empty ... 631
31.6. Sliding window after becoming full ... 631
31.7. Sliding window overwriting old elements ... 631
31.8. Delta dependencies from compressing three commits ... 636
31.9. Delta dependencies with compression size limits ... 638
32.1. Files stored end-to-end ... 644
32.2. Files stored in fixed-size blocks ... 644
B.1. Bitwise-and and bitwise-or of two numbers ... 715
B.2. Setting a `File.open` mode with bitwise-or ... 715
B.3. Checking a `File.open` mode with bitwise-and ... 716
B.4. Checking a non-matching `File.open` mode ... 716
B.5. Packing values into a byte ... 716
B.6. Decimal, hexadecimal and binary representations of numbers ... 717