
Contents

License and acknowledgements	xviii
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	3
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?	69
5.3. Reorganising the project	71
6. The index	73
6.1. The add command	73
6.2. Inspecting <code>.git/index</code>	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 <code>.git/index</code>	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	97
7.3.2. Replacing a file with a directory	99
7.3.3. Replacing a directory with a file	101
7.4. Handling bad inputs	105
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	163
10.2.1. Added files	163
10.2.2. Modified files	166
10.2.3. Deleted files	168
10.3. The long format	169
10.3.1. Making the change easy	171
10.3.2. Making the easy change	174
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	242
13.3.2. Interpreting the AST	244
13.3.3. Revisions and object IDs	248
14. Migrating between trees	254
14.1. Telling trees apart	255
14.2. Planning the changes	260
14.3. Updating the workspace	263
14.4. Updating the index	265
14.5. Preventing conflicts	266
14.5.1. Single-file status checks	267

14.5.2. Checking the migration for conflicts	269
14.5.3. Reporting conflicts	272
14.6. The perils of self-hosting	273
15. Switching branches	275
15.1. Symbolic references	277
15.1.1. Tracking branch pointers	278
15.1.2. Detached HEAD	279
15.1.3. Retaining detached histories	281
15.2. Linking HEAD on checkout	282
15.2.1. Reading symbolic references	283
15.3. Printing checkout results	284
15.4. Updating HEAD on commit	288
15.4.1. The master branch	290
15.5. Branch management	291
15.5.1. Parsing command-line options	292
15.5.2. Listing branches	294
15.5.3. Deleting branches	298
16. Reviewing history	300
16.1. Linear history	300
16.1.1. Medium format	302
16.1.2. Abbreviated commit IDs	303
16.1.3. One-line format	304
16.1.4. Branch decoration	305
16.1.5. Displaying patches	308
16.2. Branching histories	311
16.2.1. Revision lists	312
16.2.2. Logging multiple branches	313
16.2.3. Excluding branches	319
16.2.4. Filtering by changed paths	326
17. Basic merging	330
17.1. What is a merge?	330
17.1.1. Merging single commits	330
17.1.2. Merging a chain of commits	332
17.1.3. Interpreting merges	334
17.2. Finding the best common ancestor	337
17.3. Commits with multiple parents	339
17.4. Performing a merge	341
17.5. Best common ancestors with merges	343
17.6. Logs in a merging history	347
17.6.1. Following all commit parents	347
17.6.2. Hiding patches for merge commits	348
17.6.3. Pruning treesame commits	348
17.6.4. Following only treesame parents	349
17.7. Revisions with multiple parents	350
18. When merges fail	351
18.1. A little refactoring	351
18.2. Null and fast-forward merges	353
18.2.1. Merging an existing ancestor	353

18.2.2. Fast-forward merge	354
18.3. Conflicted index entries	356
18.3.1. Inspecting the conflicted repository	357
18.3.2. Stages in the index	358
18.3.3. Storing entries by stage	360
18.3.4. Storing conflicts	361
18.4. Conflict detection	363
18.4.1. Concurrency, causality and locks	364
18.4.2. Add/edit/delete conflicts	366
18.4.3. File/directory conflicts	370
19. Conflict resolution	373
19.1. Printing conflict warnings	373
19.2. Conflicted status	376
19.2.1. Long status format	377
19.2.2. Porcelain status format	378
19.3. Conflicted diffs	379
19.3.1. Unmerged paths	379
19.3.2. Selecting stages	380
19.4. Resuming a merge	382
19.4.1. Resolving conflicts in the index	382
19.4.2. Retaining state across commands	382
19.4.3. Writing a merge commit	384
20. Merging inside files	387
20.1. The diff3 algorithm	389
20.1.1. Worked example	389
20.1.2. Implementing diff3	391
20.1.3. Using diff3 during a merge	397
20.2. Logging merge commits	398
20.2.1. Unifying hunks	404
20.2.2. Diffs during merge conflicts	407
20.2.3. Diffs for merge commits	408
21. Correcting mistakes	410
21.1. Removing files from the index	410
21.1.1. Preventing data loss	411
21.1.2. Refinements to the <code>rm</code> command	414
21.2. Resetting the index state	417
21.2.1. Resetting to a different commit	420
21.3. Discarding commits from your branch	421
21.3.1. Hard reset	423
21.3.2. I'm losing my HEAD	427
21.4. Escaping from merges	428
22. Editing messages	431
22.1. Setting the commit message	432
22.2. Composing the commit message	433
22.2.1. Launching the editor	435
22.2.2. Starting and resuming merges	437
22.3. Reusing messages	440
22.3.1. Amending the HEAD	443

22.3.2. Recording the committer	445
23. Cherry-picking	449
23.1. Cherry-picking a single commit	452
23.1.1. New types of pending commit	454
23.1.2. Resuming from conflicts	456
23.2. Multiple commits and ranges	459
23.2.1. Rev-list without walking	459
23.2.2. Conflicts during ranges	461
23.2.3. When all else fails	466
24. Reshaping history	470
24.1. Changing old commits	470
24.1.1. Amending an old commit	470
24.1.2. Reordering commits	471
24.2. Rebase	473
24.2.1. Rebase onto a different branch	475
24.2.2. Interactive rebase	476
24.3. Reverting existing commits	480
24.3.1. Cherry-pick in reverse	482
24.3.2. Sequencing infrastructure	483
24.3.3. The revert command	485
24.3.4. Pending commit status	489
24.3.5. Reverting merge commits	490
24.4. Stashing changes	492
III. Distribution	496
25. Configuration	497
25.1. The Git config format	497
25.1.1. Whitespace and comments	498
25.1.2. Abstract and concrete representation	500
25.2. Modelling the .git/config file	502
25.2.1. Parsing the configuration	503
25.2.2. Manipulating the settings	506
25.2.3. The configuration stack	509
25.3. Applications	511
25.3.1. Launching the editor	511
25.3.2. Setting user details	512
25.3.3. Changing diff formatting	512
25.3.4. Cherry-picking merge commits	514
26. Remote repositories	518
26.1. Storing remote references	519
26.2. The remote command	520
26.2.1. Adding a remote	521
26.2.2. Removing a remote	523
26.2.3. Listing remotes	523
26.3. Refspecs	525
26.4. Finding objects	528
27. The network protocol	534
27.1. Programs as ad-hoc servers	534
27.2. Remote agents	536

27.3. The packet-line protocol	537
27.4. The pack format	541
27.4.1. Writing packs	542
27.4.2. Reading from packs	546
27.4.3. Reading from a stream	550
28. Fetching content	555
28.1. Pack negotiation	555
28.1.1. Non-fast-forward updates	557
28.2. The fetch and upload-pack commands	559
28.2.1. Connecting to the remote	561
28.2.2. Transferring references	562
28.2.3. Negotiating the pack	563
28.2.4. Sending object packs	566
28.2.5. Updating remote refs	567
28.2.6. Connecting to remote repositories	571
28.3. Clone and pull	572
28.3.1. Pulling and rebasing	575
28.3.2. Historic disagreement	577
29. Pushing changes	579
29.1. Shorthand refsspecs	579
29.2. The push and receive-pack commands	581
29.2.1. Sending update requests	583
29.2.2. Updating remote refs	588
29.2.3. Validating update requests	593
29.3. Progress meters	595
30. Delta compression	601
30.1. The XDelta algorithm	601
30.1.1. Comparison with diffs	604
30.1.2. Implementation	605
30.2. Delta encoding	609
30.3. Expanding deltas	612
31. Compressing packs	615
31.1. Finding similar objects	615
31.1.1. Generating object paths	618
31.1.2. Sorting packed objects	619
31.2. Forming delta pairs	622
31.2.1. Sliding-window compression	624
31.2.2. Limiting delta chain length	627
31.3. Writing and reading deltas	630
32. Packs in the database	635
32.1. Indexing packs	636
32.1.1. Extracting TempFile	637
32.1.2. Processing the incoming pack	639
32.1.3. Generating the index	640
32.1.4. Reconstructing objects	643
32.1.5. Storing the index	644
32.2. A new database backend	648
32.2.1. Reading the pack index	650

32.2.2. Replacing the backend	653
32.3. Offset deltas	658
33. Working with remote branches	660
33.1. Remote-tracking branches	660
33.1.1. Logging remote branches	660
33.1.2. Listing remote branches	662
33.2. Upstream branches	663
33.2.1. Setting an upstream branch	664
33.2.2. Safely deleting branches	669
33.2.3. Upstream branch divergence	670
33.2.4. The <code>@{upstream}</code> revision	673
33.2.5. Fetching and pushing upstream	674
34. ...and everything else	676
IV. Appendices	678
A. Programming in Ruby	679
A.1. Installation	679
A.2. Core language	680
A.2.1. Control flow	680
A.2.2. Error handling	683
A.2.3. Objects, classes, and methods	684
A.2.4. Blocks	690
A.2.5. Constants	691
A.3. Built-in data types	692
A.3.1. true, false and nil	692
A.3.2. Integer	692
A.3.3. String	693
A.3.4. Regexp	694
A.3.5. Symbol	694
A.3.6. Array	694
A.3.7. Range	696
A.3.8. Hash	697
A.3.9. Struct	698
A.4. Mixins	698
A.4.1. Enumerable	699
A.4.2. Comparable	701
A.5. Libraries	702
A.5.1. Digest	702
A.5.2. FileUtils	702
A.5.3. Forwardable	703
A.5.4. Open3	703
A.5.5. OptionParser	703
A.5.6. Pathname	703
A.5.7. Set	704
A.5.8. Shellwords	704
A.5.9. StringIO	704
A.5.10. StringScanner	705
A.5.11. Time	705
A.5.12. URI	705

Building Git

A.5.13. zlib	705
B. Bitwise arithmetic	706