

Principia Softwarica: The Version Control System `git`⁹ version 0.1

Yoann Padioleau
`yoann.padioleau@gmail.com`

with code from
Ori Bernstein

May 25, 2026

The text and figures are Copyright © 2014–2026 Yoann Padioleau.
All rights reserved.

The source code is Copyright © 2019 Ori Bernstein.
MIT license.

Contents

1	Introduction	12
1.1	Motivations	12
1.2	The version control system Git (and git9)	12
1.3	Other version control systems	13
1.4	Getting started	15
1.5	Requirements	15
1.6	About this document	15
1.7	Copyright	15
1.8	Acknowledgments	15
2	Overview	17
2.1	Version control system principles	17
2.1.1	Storing past versions	17
2.1.2	Tracking changes	18
2.1.3	Concurrent development	18
2.1.4	Parallel development	18
2.1.5	Centralized versus distributed	18
2.2	git command-line interface	18
2.3	Git concepts and data structures	19
2.3.1	Object store	19
2.3.2	References	21
2.3.3	Staging area	21
2.4	Repository format: .git/	23
2.5	hello/.git/	24
2.5.1	Creating a repository	24
2.5.2	Staging a diff	25
2.5.3	Committing a diff	25
2.5.4	Managing branches	26
2.5.5	Inspecting objects	27
2.5.6	Other commands	28
2.6	Code organization	28
2.7	Software architecture	28
2.8	Book structure	28
I	Foundations	29
3	Core Data Structures	30
3.1	Secure Hash Algorithm (SHA1) hashes	30
3.1.1	Binary hashes	30

3.1.2	Hexadecimal hashes	31
3.1.3	Conversion functions	31
3.2	Object	32
3.2.1	Object types: <code>Gxxx</code>	33
3.2.2	Object flags: <code>Cxxx</code>	33
3.2.3	Reference counting	33
3.2.4	Blobs	34
3.2.5	Trees and <code>Tinfo</code>	34
3.2.6	Commits and <code>Cinfo</code>	36
3.3	Objects	37
3.3.1	Object set: <code>Objset</code>	37
3.3.2	Object queue: <code>Objq</code>	38
3.4	<code>.git/INDEX9</code>	40
3.4.1	Status character: <code>RMAUT</code>	40
3.4.2	<code>Idxent</code>	41
3.4.3	<code>idx</code>	41
4	Reading from a Repository	43
4.1	Objects	43
4.1.1	<code>readobject()</code>	43
4.1.2	<code>readidxobject()</code> part 1	43
4.1.3	<code>readloose()</code> and <code>Cloaded</code>	44
4.1.4	<code>parseobject()</code> and <code>Cparsed</code>	45
4.1.5	Decompression	46
4.1.6	Reading a blob	47
4.1.7	Reading a tree	47
4.1.8	Reading a commit	48
4.2	References	50
4.2.1	<code>readref()</code>	50
4.2.2	<code>.refs/</code> prefixes	51
4.2.3	<code>matchpfx()</code>	51
4.3	Index	52
5	Writing to a Repository	55
5.1	Objects	55
5.1.1	<code>Objbuf</code>	55
5.1.2	<code>writeobj()</code>	55
5.1.3	Compression	56
5.1.4	Writing a blob	56
5.1.5	Writing a tree	57
5.1.6	Writing a commit	58
5.2	References	59
5.3	Index	59
6	git/conf	60
6.1	Repository configuration: <code>.git/config</code>	60
6.2	User configuration: <code>~/lib/git/config</code>	60
6.3	Usage	60
6.4	<code>main()</code>	60
6.5	<code>gitinit()</code> and <code>findrepo()</code>	61

6.6	git/conf -r, printing the project root	62
6.7	git/conf <key>, looking up a key	62
6.8	Advanced features	63
6.8.1	git/conf -a, showing all values	63
6.8.2	git/conf -f <config>, non-standard config file path	64
7	git/fs	65
7.1	Introduction	65
7.2	Usage	65
7.3	main()	65
7.4	Data structures	66
7.4.1	Qxxx and qroot	66
7.4.2	Gitaux and Crumb	67
7.4.3	Objlist	67
7.4.4	Uqid and Cache	68
7.5	gitxxx() skeletons	69
7.5.1	gitattach() and gitdir	69
7.5.2	gitclone() and gitdestroyfid()	70
7.5.3	gitopen()	71
7.5.4	gitstat()	72
7.5.5	gitread()	72
7.5.6	gitwalk1()	73
7.6	.git/fs/	74
7.6.1	.git/fs/ctl	75
7.6.2	.git/fs/object/	76
7.6.3	.git/fs/object/<blobid>	78
7.6.4	.git/fs/object/<treeid>/	78
7.6.5	.git/fs/object/<commitid>/	80
7.6.6	.git/fs/object/<commitid>/tree	81
7.6.7	.git/fs/object/<commitid>/parent	82
7.6.8	.git/fs/object/<commitid>/msg	82
7.6.9	.git/fs/object/<commitid>/hash	83
7.6.10	.git/fs/object/<commitid>/author	83
7.6.11	.git/fs/object/<commitid>/committer	83
7.6.12	.git/fs/HEAD/	83
7.6.13	.git/fs/branch/	84
7.7	git/fs -m, non-standard mount point	86
8	git/query	87
8.1	Usage	87
8.2	Commit expressions	87
8.3	main()	87
8.4	Data structures	88
8.4.1	Eval	88
8.5	resolverefs()	89
8.6	resolveref()	89
8.7	Commit expression evaluator	89
8.7.1	evalpostfix()	90
8.7.2	word()	92
8.7.3	push() and pop()	92

8.7.4	<code>parent()</code> and <code>^</code>	93
8.7.5	<code>lca()</code> and <code>@</code>	93
8.7.6	<code>range()</code> and <code>'..'</code>	94
8.8	<code>paint()</code>	95
8.9	Advanced features	98
8.9.1	<code>git/query -c</code> , list changes between commits	98
8.9.2	<code>git/query -p</code> , full path listing	101
8.9.3	<code>git/query -r</code> , reversed listing	101
9	<code>git/walk</code>	102
9.1	Introduction	102
9.2	Usage	102
9.3	<code>main()</code>	102
9.4	Initializations	104
9.4.1	The index state: <code>idx</code>	104
9.4.2	The working directory state: <code>wdir</code>	104
9.5	Walking	107
9.6	Showing	108
9.7	Finalizations	111
9.8	Advanced features	112
9.8.1	<code>git/walk -b <base></code>	112
9.8.2	<code>git/walk -I</code>	112
9.8.3	<code>git/walk -r</code>	113
9.8.4	<code>git/walk -c</code>	114
10	<code>git/save</code>	115
10.1	Usage	115
10.2	<code>main()</code>	115
10.3	<code>findroot()</code>	117
10.4	<code>treeify()</code>	118
10.5	<code>mkcommit()</code>	120
10.6	Advanced features	120
10.6.1	<code>git/save -p</code> , multi-parents (merge) commits	120
10.6.2	<code>git/save -d</code> , non-standard commit date	120
II	Basic Commands	122
11	Creating a Repository	123
11.1	Initializing a new repository: <code>git init</code>	123
11.1.1	<code>init.rc</code>	123
11.1.2	Usage	123
11.2	Copying a local repository: <code>cp -r</code>	124
11.3	Cloning a remote repository: <code>git clone</code>	124
12	Staging a Diff	125
12.1	Adding files: <code>git add</code>	125
12.1.1	Usage	125
12.1.2	<code>add.rc</code>	125
12.1.3	<code>gitup()</code> part 1	125

12.1.4	<code>drop()</code>	126
12.2	Removing files: <code>git rm</code>	126
13	Committing a Diff	127
13.1	Committing the index: <code>git commit</code>	127
13.1.1	Usage	127
13.1.2	<code>commit.rc</code>	127
13.1.3	<code>whoami()</code>	128
13.1.4	<code>gitup()</code> part 2	128
13.1.5	<code>findbranch()</code>	128
13.1.6	<code>parents()</code>	129
13.1.7	<code>editmsg()</code>	129
13.1.8	<code>commit()</code>	130
13.1.9	<code>update()</code>	130
13.2	<code>git/walk -fRMA</code>	131
13.3	<code>git/query \$branch</code>	131
13.4	<code>git/save</code>	131
13.5	Advanced features	131
13.5.1	<code>git/commit -r</code> , revising a commit	131
13.5.2	<code>git/commit -p</code> , partial commit and hunks selection	131
14	Branching	133
14.1	<code>git branch</code>	133
14.1.1	Usage	133
14.1.2	<code>branch.rc</code>	133
14.2	Listing branches: <code>git branch</code>	133
14.3	Creating a branch: <code>git branch <name></code>	134
14.4	Deleting a branch: <code>git branch -d <name></code>	136
14.5	Checking out a branch: <code>git checkout</code>	136
14.5.1	Computing the index (and work tree) from trees	136
14.5.2	Creating a file from a blob	136
14.6	Resetting a branch: <code>git reset</code>	136
15	Merging Branches	137
15.1	merging branches: <code>git merge</code>	137
15.1.1	Usage	137
15.1.2	<code>merge.rc</code>	137
15.1.3	<code>merge()</code>	138
15.1.4	<code>merge1()</code>	138
15.1.5	<code>mergeperm()</code>	138
15.2	Merging two trees	139
15.3	Merging two files	139
15.4	<code>diff3</code> and <code>merge3</code>	139
15.5	Merging conflicts	139
15.6	Committing merged branches: <code>git commit</code>	139
16	Inspecting	140
16.1	Showing the commit history: <code>git log</code>	140
16.1.1	Usage	140
16.1.2	<code>main()</code>	140

16.1.3	<code>showcommits()</code>	141
16.1.4	<code>show()</code>	142
16.1.5	Advanced features	143
16.2	Representing changes	146
16.2.1	Tree changes	146
16.2.2	File changes	146
16.3	Showing file differences: <code>git diff</code>	146
16.3.1	Usage	146
16.3.2	<code>diff.rc</code>	147
16.3.3	<code>diff -u</code>	148
16.3.4	Computing tree changes	148
16.3.5	Computing file changes	148
16.3.6	Showing changes	148
17	Packing and Unpacking	149
17.1	Packing objects: <code>git repack</code>	149
17.1.1	Usage	149
17.1.2	<code>main.c</code>	149
17.1.3	<code>listrefs()</code>	150
17.1.4	<code>cleanup()</code>	151
17.2	Data structures	152
17.2.1	<code>Packf</code> and <code>packf</code>	152
17.2.2	Pack Index	152
17.2.3	<code>Meta</code> and <code>Metavec</code>	152
17.2.4	<code>Delta</code>	153
17.2.5	<code>Dtab</code> and <code>Dblock</code>	154
17.3	Searching the pack index	155
17.4	<code>readidxobject()</code> part 2	157
17.5	Reading packs	158
17.5.1	Refreshing packs and loading indexes	158
17.5.2	Opening a pack	160
17.5.3	Reading a packed object	161
17.6	Writing packs	163
17.6.1	<code>writepack()</code>	163
17.6.2	<code>readmeta()</code>	163
17.6.3	<code>pickdeltas()</code>	166
17.6.4	<code>genpack()</code>	169
17.7	Indexing packs	173
17.7.1	<code>indexpack()</code>	173
17.8	<code>Delta</code>	177
17.8.1	<code>Odelta</code>	177
17.8.2	<code>Rdelta</code>	179
18	Exchanging Objects	181
18.1	Upstream and remote origin	181
18.2	Client/server architecture	181
18.2.1	<code>Conn</code>	181
18.3	Pulling updates from another repository: <code>git pull</code>	182
18.3.1	Usage	182
18.3.2	<code>pull.rc</code>	182

18.3.3	update()	182
18.3.4	merge	183
18.3.5	git/branch -mnb <remote> <local>	184
18.4	git/get	184
18.4.1	Usage	184
18.4.2	main()	184
18.4.3	gitconnect() part 1	185
18.4.4	closeconn()	185
18.4.5	localrepo() and servelocal()	185
18.4.6	fetchpack()	186
18.4.7	Advanced features	196
18.5	Pushing changes to another repository: git push	196
18.5.1	Usage	196
18.5.2	push.rc	197
18.6	git/send	198
18.6.1	Usage	198
18.6.2	main()	198
18.6.3	sendpack()	198
18.6.4	Advanced features	202
18.7	Cloning a remote repository: git clone	203
18.7.1	Usage	203
18.7.2	clone.rc	203
19	Networking	206
19.1	gitconnect() part 2	206
19.2	git:// protocol	207
19.2.1	dialgit()	207
19.2.2	githandshake()	208
19.3	Packet IO	208
19.3.1	Writing packets	208
19.3.2	Reading packets	209
19.4	Capabilities	210
19.4.1	Conn Capabilities	210
19.4.2	parsecaps()	210
19.5	git:// client	211
19.6	git:// server	211
19.6.1	Usage	211
19.6.2	main.c	211
19.6.3	git-upload-pack	213
19.6.4	git-receive-pack	215
19.6.5	Advanced features	222
III	Beyond the Basics	223
20	Core Algorithms	224
20.1	SHA1	224
20.2	Murmurhash2	224
20.3	Zip and Unzip	225
20.4	Diff	225

20.4.1	Overview	225
20.4.2	Data structures	225
20.4.3	Entry point: <code>Diffs.diff()</code>	225
20.4.4	Edit distance basic algorithm ($O(n^2)$ in space/time)	225
20.5	Diff3	225
20.5.1	Overview	225
20.5.2	Data structures	225
20.5.3	Entry point: <code>diff3()</code>	225
20.5.4	Displaying merging conflicts	225
21	Supporting Programs	226
21.1	<code>diff</code>	226
21.1.1	Usage	226
21.1.2	Data structures	226
21.1.3	<code>main()</code>	226
21.1.4	Advanced features	227
21.2	<code>patch</code>	227
21.2.1	Usage	227
21.2.2	Data structures	228
21.2.3	<code>main()</code>	228
21.2.4	Advanced features	229
21.3	<code>merge3</code>	229
21.3.1	Usage	229
21.3.2	Data structures	229
21.3.3	<code>main()</code>	229
22	Exchanging Patches	231
22.1	<code>git/export</code>	231
22.1.1	Usage	231
22.1.2	<code>export.rc</code>	231
22.2	<code>git/import</code>	233
22.2.1	Usage	233
22.2.2	<code>import.rc</code>	233
23	Advanced Features	236
23.1	Tracking symbolic links	236
23.2	Splitting a large repository: submodules	237
24	Advanced Commands	238
24.1	Tagging a commit with a name	238
24.1.1	Reading a tag	238
24.1.2	Writing a tag	238
24.2	<code>git/hist</code>	238
24.2.1	Usage	238
24.2.2	<code>hist.rc</code>	239
24.3	Reverting a change: <code>git revert</code>	239
24.3.1	Usage	239
24.3.2	<code>revert.rc</code>	239
24.4	Rebase: <code>git rebase</code>	240
24.4.1	Usage	240

24.4.2	rebase.rc	240
25	Advanced Networking	242
25.1	Other clients	242
25.1.1	http://	242
25.1.2	ssh://	245
25.1.3	gits://	246
26	Advanced Topics	248
26.1	Bytes versus ASCII versus UTF-8	248
26.2	Reliability and signals	248
26.3	Optimizations	248
26.3.1	Object cache	248
26.4	Fast import and export	250
26.5	Alternates	250
26.6	Other repository format	250
26.6.1	Bare repository	250
26.6.2	.git file	250
26.7	Security	250
27	Conclusion	251
A	Debugging	252
A.1	Format dumpers	252
A.2	git/fs -d	253
A.3	git/xxx -d	253
A.4	Tracing packets	254
B	Error Management	255
C	Utilities	256
C.1	Error malloc	256
C.2	String utilities	258
C.3	File and directory utilities	260
D	Extra Code	261
D.1	git9/	261
D.1.1	git9/common.rc	261
D.1.2	git9/add.rc	261
D.1.3	git9/branch.rc	261
D.1.4	git9/clone.rc	261
D.1.5	git9/commit.rc	261
D.1.6	git9/compat.rc	261
D.1.7	git9/diff.rc	266
D.1.8	git9/export.rc	266
D.1.9	git9/import.rc	266
D.1.10	git9/hist.rc	266
D.1.11	git9/init.rc	266
D.1.12	git9/merge.rc	266
D.1.13	git9/pull.rc	266
D.1.14	git9/push.rc	266

D.1.15	git9/rebase.rc	266
D.1.16	git9/revert.rc	266
D.1.17	git9/rm.rc	266
D.1.18	git9/conf.c	266
D.1.19	git9/delta.c	266
D.1.20	git9/fs.c	267
D.1.21	git9/get.c	269
D.1.22	git9/git.h	270
D.1.23	git9/log.c	272
D.1.24	git9/objset.c	273
D.1.25	git9/ols.c	273
D.1.26	git9/pack.c	276
D.1.27	git9/proto.c	279
D.1.28	git9/query.c	280
D.1.29	git9/ref.c	281
D.1.30	git9/repack.c	282
D.1.31	git9/save.c	282
D.1.32	git9/send.c	283
D.1.33	git9/serve.c	283
D.1.34	git9/util.c	284
D.1.35	git9/walk.c	285
D.2	diff/	287
D.2.1	diff/diff.c	287
D.2.2	diff/diff.h	287
D.2.3	diff/diffdir.c	288
D.2.4	diff/diffio.c	291
D.2.5	diff/diffreg.c	299
D.2.6	diff/merge3.c	306
D.2.7	diff/util.c	310
D.3	misc/	312
D.3.1	misc/patch.c	312

Glossary **326**

Index **327**

References **340**

Chapter 1

Introduction

The goal of this book is to explain with full details the source code of a version control system.

1.1 Motivations

Why a version control system (VCS)? Because I think you are a better programmer if you fully understand how things work under the hood, and a VCS is one of the tools a programmer uses the most. Indeed, programmers use VCSs—which manage efficiently changes to a file or set of files—to manage the vital product of their labour: source code.

VCSs allow to store and retrieve past versions of a file, and to record who made each change, when, where, and why. VCSs are mostly used by programmers but you can use a VCS to manage any text-based documents (e.g., the source of this book), configuration files, and even binaries.

For projects with a single developer, a VCS is useful to keep track of changes: it allows the programmer to easily go back to past versions if he messed things up. For projects with multiple developers, a VCS is almost mandatory: it allows programmers to collaborate with each other and even to work on and modify the same files concurrently.

Here are a few questions I hope this book will answer:

- What are the fundamental concepts of a VCS? What are the core algorithms behind a VCS?
- How does a VCS store efficiently multiple versions of a file?
- How does a VCS represent changes to a file, or changes to a set of files? How does a VCS represent the addition, deletion, or renaming of a file?
- How does a VCS allow multiple users to work on and modify the same files at the same time? How can a VCS help coordinate the actions of multiple users?
- How does a VCS support parallel developments in a project? How can some developers work on the main release, some developers on experimental features, and others on fixing bugs on previously shipped releases all at the same time? What is a branch? How does a VCS reconcile multiple branches?
- What is a merge algorithm? When is a merge safe? What is a merge conflict? How does a VCS resolve conflicts?

1.2 The version control system Git (and git9)

There are many open source VCSs (Subversion, Mercurial, etc.) with different user interfaces, storage strategies, concurrency models, or features. I will present a few of those VCSs in Section 1.3. For this book, I decided

to base the presentation on Git¹ because it is one of the most popular VCSs in the open source community. Moreover, when coupled with the hosting website GitHub², Git makes it really easy for people to collaborate with each other.

However, Git is a rather large project with more than 200 000 LOC. It is impossible to present all this code in a book of a reasonable size. There are a few clones of Git written in higher-level languages than C, for example, Dulwich³ written in Python with only 16 000 LOC. I could have based this book on Dulwich, but this would introduce another language in the Principia Softwarica book series in addition to C. This would also in turn require to present the code of the Python interpreter, which contains unfortunately more than 170 000 lines of C code (almost the size of all the Plan 9 programs together).

Fortunately, in 2021 Ori Bernstein decided to rewrite Git for Plan 9 and redesigned the program to use far less code. This is why I will explain in this book the code of `git9`⁴, which contains about 10 000 lines LOC; most of the code is written in C but about 1000 LOC are written in `rc`, the Plan 9 shell (see the SHELL book [Pad18]).

Like for most books in Principia Softwarica, I chose a Plan 9 program because those programs are simple, small, elegant, open source, and they form together a coherent set. Note that `git9` was not part of the original Plan 9 but `git9` follows the Plan 9 philosophy and coding style.

1.3 Other version control systems

Here are a few VCSs that I considered for this book, but which I ultimately discarded:

- The Revision Control System (RCS) [Tic85]⁵ was the first open source VCS. Like its predecessor, the Source Code Control System (SCCS) [Roc75]⁶, RCS uses *deltas* to store efficiently the past versions of a file (Section 20.4 will present the Diff algorithm to compute those deltas). However, RCS improves over SCCS by using *reverse-deltas* to enable the user to also retrieve quickly the last version of a file (a recurring operation).

RCS is still a reasonable choice to manage a small project with a single developer or a small set of developers who can share access to a common directory (for example, by using NFS and symbolic links). However, the use of *locks* in RCS to forbid multiple developers to modify the same file at the same time and its focus on individual files make RCS inadequate for large projects with many independent developers. Moreover, even if RCS is very limited compared to modern VCSs like Git, RCS still contains more than 17 500 LOC.

- The Concurrent Versions System CVS [Gru86]⁷ was the most popular VCS for over a decade before distributed VCSs (e.g., Git) took over. CVS introduced a few important innovations. First, CVS was designed to operate on a set of files at once⁸, with files possibly organized in a tree. With CVS, you can group together related changes to a set of files and you can easily retrieve past versions of a whole project. Secondly, CVS allows multiple users to work on and modify the same files *concurrently* (hence the 'C' in CVS). CVS relies on the `merge` program instead of locks (Section 20.5 will present the Diff3 algorithm which underlies the `merge` program). Finally, CVS introduced later in 1995 a *client/server* architecture where a *repository* could be stored on a remote machine. When coupled with the free hosting site Sourceforge⁹, CVS allowed developers to easily start and collaborate on new projects.

CVS started as a few shell scripts using RCS as a backend. However, today CVS is a very large C program with more than 80 000 LOC. The backend file `src/rcs.s` contains already 9000 LOC.

¹<https://git-scm.com/>

²<https://github.com>

³<https://www.dulwich.io/>

⁴<https://github.com/oridb/git9>

⁵<https://www.gnu.org/software/rcs/>

⁶<http://sccs.sourceforge.net/>

⁷<http://www.nongnu.org/cvs/>

⁸RCS can operate on multiple files by using shell wildcards, but it was not designed for it.

⁹<https://sourceforge.net>

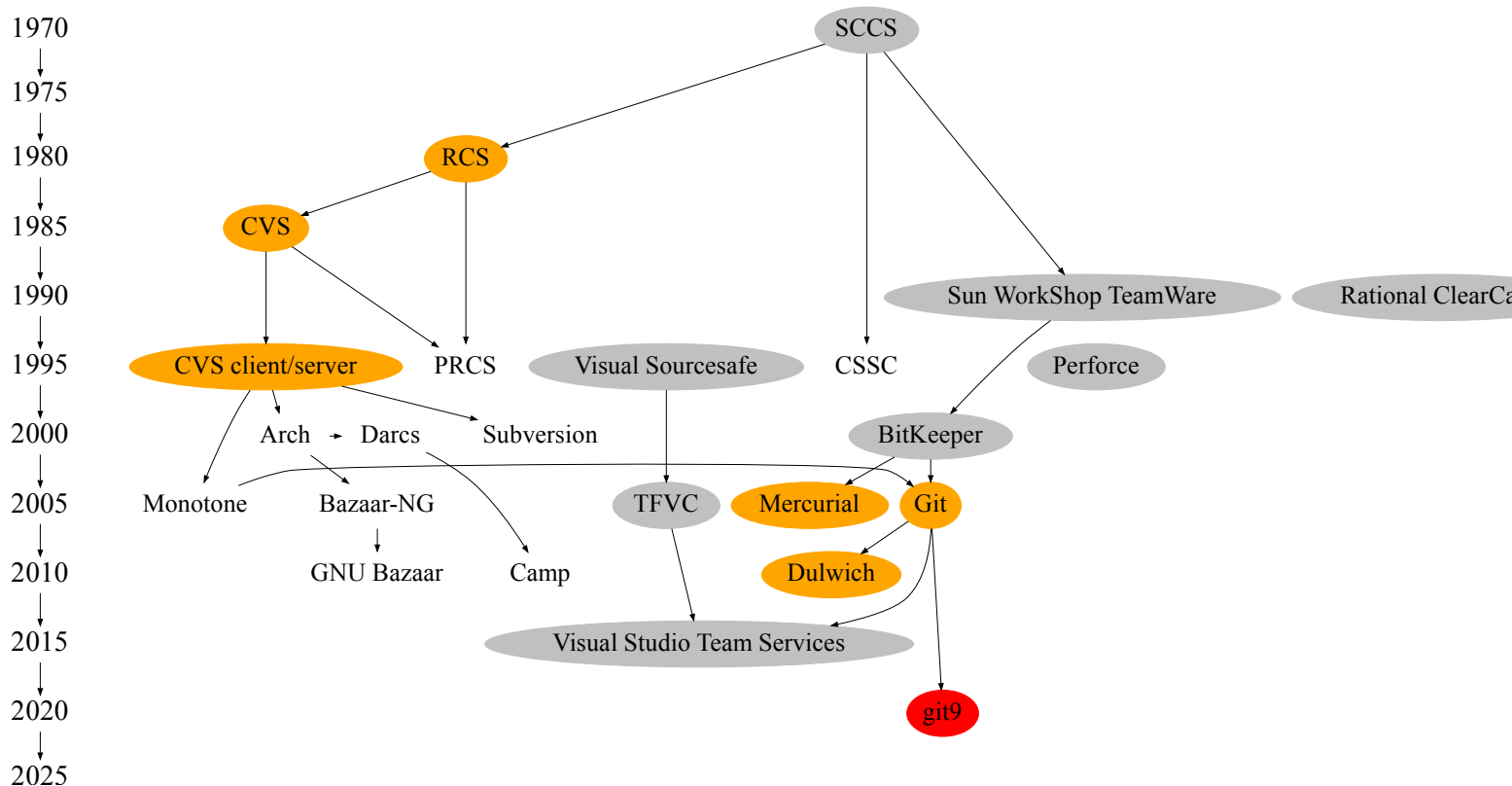


Figure 1.1: Version control systems timeline

- Git [CS14]¹⁰ was originally created by Linus Torvalds in 2005 to manage the source code of the Linux kernel. It followed a series of *distributed VCSs* (e.g., TeamWare, BitKeeper, Arch, Darcs, Monotone) designed to overcome the main limitations of CVS (a centralized VCS). Because a distributed VCS (DVCS) does not require access to a central repository, it allows developers to work independently offline. A DVCS makes it also easy and cheap to create *branches* representing parallel developments, and to reconcile later those branches (see Section 2.1.5 for more discussions on the advantages of distributed VCSs over centralized VCSs).

As I explained in Section 1.2, Git is a large program with more than 200 000 LOC spread over more than 400 files (not including the tests, GUIs, and extra contributions). The first version of Git was small and contained only 1000 LOC¹¹. However, it contained only a few low-level commands that were not easy to use. This forced programmers to develop and use the extra program *cogito*¹², which provided an extra layer of higher-level commands (called *porcelain*), but added many more lines of code.

- Mercurial [O'S09]¹³ is another popular DVCS started also in 2005 by another Linux kernel programmer: Matt Mackal. It is mostly written in Python and relies only on a few C files for critical operations. Its code is arguably easier to understand than Git but it still contains more than 100 000 LOC spread over more than 170 files (not including the tests, extensions, and extra contributions).

Figure 1.1 presents a timeline of major VCSs. *git9* is not as efficient and complete as Git or Mercurial, but it provides almost all of the essential features of those programs with more than an order of magnitude less code.

¹⁰<https://git-scm.com>

¹¹<https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290>

¹²<http://git.or.cz/cogito/>

¹³<https://www.mercurial-scm.org/>

1.4 Getting started

To play with `git9`, you will first need to install the Plan 9 fork used in Principia Softwarica (see <https://www.principia-softwarica.org/getting-started.html>). Once installed, you can test `git9` under Plan 9 with the following commands:

```
1 $ cd /tests/  
2 $ mkdir hello  
3 $ cd hello/  
4 $ git/init  
5 ...  
6 $ echo "Hello Git" > hello.txt  
7 $ git/add hello.txt  
8 $ git/commit -m "first commit" .  
9 heads/master: 56adf1865
```

The command Line 4 initializes a new *repository* by creating appropriate metadata in the `/tests/hello/.git/` subdirectory. Line 7 then adds the new file `hello.txt` to the *staging area* of Git (I will explain later in Section 2.3.3 what is a staging area). Finally, Line 8 *commits* what was staged to the repository and records this commit with the message “first commit”.

1.5 Requirements

Because most of this book is made of C source code, you will need a good knowledge of the C programming language [KR88] to understand it. A significant part of `git9` is made of `rc` shell scripts so you will need to read the SHELL book [Pad18] or one of the article introducing `rc` (e.g., `shells/docs/rc.pdf`). Moreover, those scripts are often using utilities such as `awk` or `sed` so reading the UTILITIES book [Pad25] would also be useful.

You do not need to know Git, or more generally any VCSs, to understand this book. However, if while reading this book you have specific questions on the interface of Git, I suggest you to consult the man pages of Git¹⁴, or any of the books on Git (e.g., [CS14]).

1.6 About this document

This document is a *literate program* [Knu92]. It derives from a set of files processed by a tool, `syncweb` [Pad09], generating either this book or the actual source code of the program. The code and its documentation are thus strongly connected.

1.7 Copyright

Most of this document is made of source code from Ori Bernstein, so those parts are copyright by him. The prose is mine and is licensed under the All Rights Reserved License.

1.8 Acknowledgments

I would like to acknowledge first the author of `git9`, Ori Bernstein, who wrote in some sense most of this book.

I would like also to thank Linus Torvalds, the original author of Git, for designing this simple but powerful VCS. Many of his design decisions seem obvious in retrospect but they are not: most VCSs chose different designs

¹⁴<https://git-scm.com/docs>

(for the storage, file permissions, for the way to handle renames, etc.) which in the end are more complicated and usually less efficient than in Git.

Chapter 2

Overview

Before showing the source code of `git9` in the following chapters, I first give an overview in this chapter of the general principles of a VCS. I also quickly describe the command-line interface of `git` (and `git9`), the format of a Git repository, and I use a simple terminal session containing `git` commands to illustrate its major features. Finally, I define terms, explain how the code is organized, and more generally give the background necessary to understand the code I will show later.

2.1 Version control system principles

The main requirements for a modern VCS are to be able to store past versions of a set of files, to track the changes to those files, and to help multiple developers to collaborate with each other on those files by supporting concurrent and parallel development (I will explain soon the differences between those two kinds of development). The next sections will each give more details about those requirements. I will also try to present the major techniques used by popular VCSs to satisfy those requirements. Finally, I will discuss two different architectures for VCSs: centralized and distributed.

2.1.1 Storing past versions

The first requirement of a VCS is to store past versions of a file, hence the 'V' in VCS. However, there is also an 'S' in VCS, which means it must be done in a systematic way. Indeed, one way to understand the usefulness of a VCS is to see how people are struggling when they are managing documents that evolve (e.g., source code, \LaTeX files, notes) without a VCS. In that situation, people often use `cp` to *backup* files, but it is inefficient space-wise and error-prone.

The repository

Working copy

Delta

Compression

2.1.2 Tracking changes

The commit

Version identifier

Change granularity

diff

patch

2.1.3 Concurrent development

Locks

Merging

Merge conflicts

2.1.4 Parallel development

Branch

Merging

A direct acyclic graph of versions

2.1.5 Centralized versus distributed

2.2 git command-line interface

I just described the general principles of a VCS, and illustrated some of those principles with examples from RCS, CVS, or Git. I will now focus exclusively on Git and give more details about the interface of its command-line program: `git`.

The command-line interface of `git` is pretty simple:

```
1 $ git
2 usage: git <init|add|rm|commit|branch|checkout|reset|...> [options]
3 $ git init
4 Initialized empty Git repository in /tests/hello/.git/
5 $ git add hello.txt
6 $ git commit -m "first commit" --author "pad <todo@todo>"
```

`git` takes first a *command* as its first argument and then options and extra arguments specific to this command. For example, the command `init` at Line 3 does not need any extra arguments, but `add` at Line 5 requires at least the name of a file or directory, and `commit` at Line 6 usually requires command-line flags. I will gradually describe the different `git` commands, their options, and implementations in the following chapters.

Note that in most of this chapter I will use `git` and not `git9` commands to be more general. The reader can thus easily test those commands on Linux or macOS by using the regular `git` program. It is easy to adapt

those commands to `git9` on Plan 9. The main difference is that `git9` accepts commands separated by a `/` (as in `git/init` or `git/add foo.txt`) instead of space.

2.3 Git concepts and data structures

To understand how Git is implemented, and to some degree to learn how to use Git effectively, you need to know the few concepts underlying Git. Those concepts are the *object store*, the *reference*, and the *staging area*. They correspond also to `git9`'s main data structures, which I will describe fully in Chapter 3. The following sections will just give an overview of those data structures.

2.3.1 Object store

To really understand Git, you need to realize that at its core, Git is a simple *content-addressable storage system*¹. Given the *hash* of a content (Git uses SHA1 hashes, as explained below), Git can retrieve back quickly the full corresponding content. Git internally is simply a *persistent hash table*, also known as a *key/value store*.

In this store, Git manipulates mainly three kinds of values, known as *objects* in Git terminology (hence the term *object store*):

- A *blob*: to represent the content of a file at a specific time.
- A *tree*: to assign names to blobs or other subtrees
- A *commit*: to associate to a specific toplevel tree a message, an author, a date, and zero or more *parent* commits

You will see another kind of objects in Section 24.1 with the *tag*. In addition to those objects, Git maintains also *references* to specific commits and an *index* of specific blobs, as explained in the next sections.

Regarding the keys of the store, Git uses the SHA1 algorithm [sha93, EJ01] to compute the hash of an object. This algorithm associates an almost unique number of 160 bits (which amounts to 20 bytes, or 40 digits in hexadecimal format) to any content of arbitrary length. Section 20.1 presents the code of this algorithm and gives more information on SHA1. For example, given the content `Hello Git\n`, the SHA1 algorithm will return the number `9f4d96d5b00d98959ea9960f069585ce42b1349a` in hexadecimal format. Here the hashing is not very interesting because the hash is bigger than the original content, but in practice most files under a VCS are far bigger than 20 bytes.

SHA1 is a complex cryptographic hash function. It is outside the scope of this book to explain how and why it works, but what is important for Git is that SHA1 is an almost perfect hash function: given two different content, there is an almost zero probability that SHA1 will return the same hash number. Such an event is called a *collision*, and in practice it should never happen. Thanks to this almost perfect hash function, Git can identify any content of any length with just a 20 bytes number. In Git, the SHA1 of a file is similar to the *inode* of a file in a filesystem (see the KERNEL book [Pad14] for more information on filesystems and inodes)², except it references a specific version of a file. In the tree object, which associates names to blobs, Git uses the SHA1 of a blob to identify this blob. In the same way, in the commit object, Git uses the SHA1 of a tree to identify the toplevel tree a commit refers to.

Figure 2.1 presents the Git objects (and their relationships) of the Git repository created in Section 1.4 after the addition of another commit. The first commit, which added `hello.txt`, is at the top left of Figure 2.1. The second commit, below, added the two files `foo.txt` and `bar.txt` under a directory `dir1/`, and also modified `hello.txt`. I will describe soon in Section 2.4 how Git stores those objects on the disk.

¹Torvalds himself introduced Git as “the stupid content tracker”.

²The author of Git, Linus Torvalds, is also the author of Linux, which explains why he chose a design inspired by filesystems.

```

commit 19d977...-----+
|tree:                |   tree 2f092e...-----+       blob 9f4d96...+
| 2f092e... -----+-->| hello.txt 9f4d96.---+----->|Hello Git   |
|parents:             |   +-----+                               |           |
| None                |                               |           |
|Author: pad          |                               +-----+
| <todo@todo>        |                               |           |
|Date: Fri Sep 29    |                               blob 557db0...+
| 14:04:46 2017 -0700|   +-->|Hello World |
|Message:            |   |   |
| first commit       |   +-->tree 0c4b27...-----+ |   |
+-----+           |   ++ dir1 7e8bb2...   | |   +-----+
                   |   || hello.txt 557db0...++
                   |   |+-----+
commit 5d9dfe...---+---+| |
|tree:                |   |   |                               blob 452a53...+
| 0c4b27...-----+---+|   |   |                               |baaaaaaaaaa | |
|parents:             |   |   |                               +>|aaaaaaaaaaa |
| 19d977-----+     |   |   |                               | |aaaaaaaaaar |
|Author: pad          |   |   |   tree 7e8bb2...-----+ | +-----+
| <todo@todo>        |   +-->| bar.txt 452a53.---+---+
|Date: Fri Sep 29    |   |   |   | foo.txt 96ea4a.---+---+ blob 96ea4a...+
| 14:26:23 2017 -0700|   +-----+ |   | foooooooooo | |
|Message:            |   +>| oooooooooooo |
| second commit, sub|   |   |   | oooooooooooo |
+-----+           |   +-----+

```

Figure 2.1: Git objects relationships for hello/.git after two commits.

Commit objects are linked together, as shown in Figure 2.1 with the second commit linked to the first commit through the `parents` field of the object. Those links represent the history of the repository, also known as its *log*. A commit can have multiple parents in the case of a *merge* (I will explain in Section 15.1 how to merge branches in Git). Thus, the log forms a *graph*, or more specifically a *direct acyclic graph* (DAG). Figure 2.2 shows such a graph for an additional series of commits following the first two commits of Figure 2.1. In this example, a developer *forked* first an experimental branch called `branch1` (I will explain briefly how to create a branch in Section 2.5.4 and fully in Section 14.3) following the second commit of Figure 2.1. At the same time, development on the main branch, called the *master branch* in Git terminology, continued and saw two commits: `commit4` and `commit6`. The experimental branch got *merged* later at `commit7`. This is why this commit has two parents. Later on, a developer created another branch called `experiment` that remained active and did not get merged yet; it follows in parallel the development of the master branch.

2.3.2 References

As I mentioned briefly in the previous section, Git maintains, in addition to the object store, *references* to specific commit objects. Those references have a *name* and a *content*. For example, in Figure 2.2, the reference named `refs/heads/experiment` points to the last commit of the `experiment` branch, and its content is the SHA1 of the `commit9` object. I will explain in Section 2.4 how Git stores those references on the disk.

Git maintains also a special reference called the *HEAD* which points to the last commit of the *current branch* (see the bottom of Figure 2.2). Its content is usually not a SHA1 but the name of another reference. By default, *HEAD* points to the master branch, as shown at the bottom of Figure 2.2, and so its content is the string `ref: refs/heads/master` (and `refs/heads/master` contains the SHA1 of the last commit of the master branch). When you switch branch (I will explain how to switch branch briefly in Section 2.5.4 and fully in Section 14.5), the content of the *HEAD* will change.

A reference pointing to the last commit of a branch is called a *head* in Git terminology and starts with `refs/heads/`. In Figure 2.2, those heads are `refs/heads/experiment` and `refs/heads/master`.

Git maintains a final set of special references called the *remotes*, for example `refs/remotes/origin/master` pointing to `commit8` in Figure 2.2. Those references are used when people are collaborating with each other and I will explain them in Chapter 18.

2.3.3 Staging area

The last important concept of Git is the *staging area*. Git operates mainly on three different areas, as shown at the top of Figure 2.3:

1. The *working copy* contains the current state of the files
2. The *repository* contains all past versions of all the files including the state of the files at the last commit
3. The *staging area* contains what the user wants to commit next

After you modify a set of files in your working copy, for example, `bar.txt`, `foo.txt`, and `misc.txt` at the right of Figure 2.3, you need to indicate which of those files and modifications should be part of the next commit. To do so, you need to *mark* those files by using the command `git add` (or `git rm` if you want to mark for deletion a file).

Once you marked all the relevant files, you can use the command `git commit` to commit the modifications to the repository. For example, in Figure 2.3, only the modifications to `foo.txt` and the deletion of `misc.txt` will be part of the next commit, and not the modifications to `bar.txt`.

Once you add a file in a repository, the file is said to be *tracked* in Git terminology. You can sometimes avoid manipulating explicitly the staging area by using instead the command `git commit -a` to automatically commit all the modifications to the tracked files. However, the staging area and `git add` give more flexibility

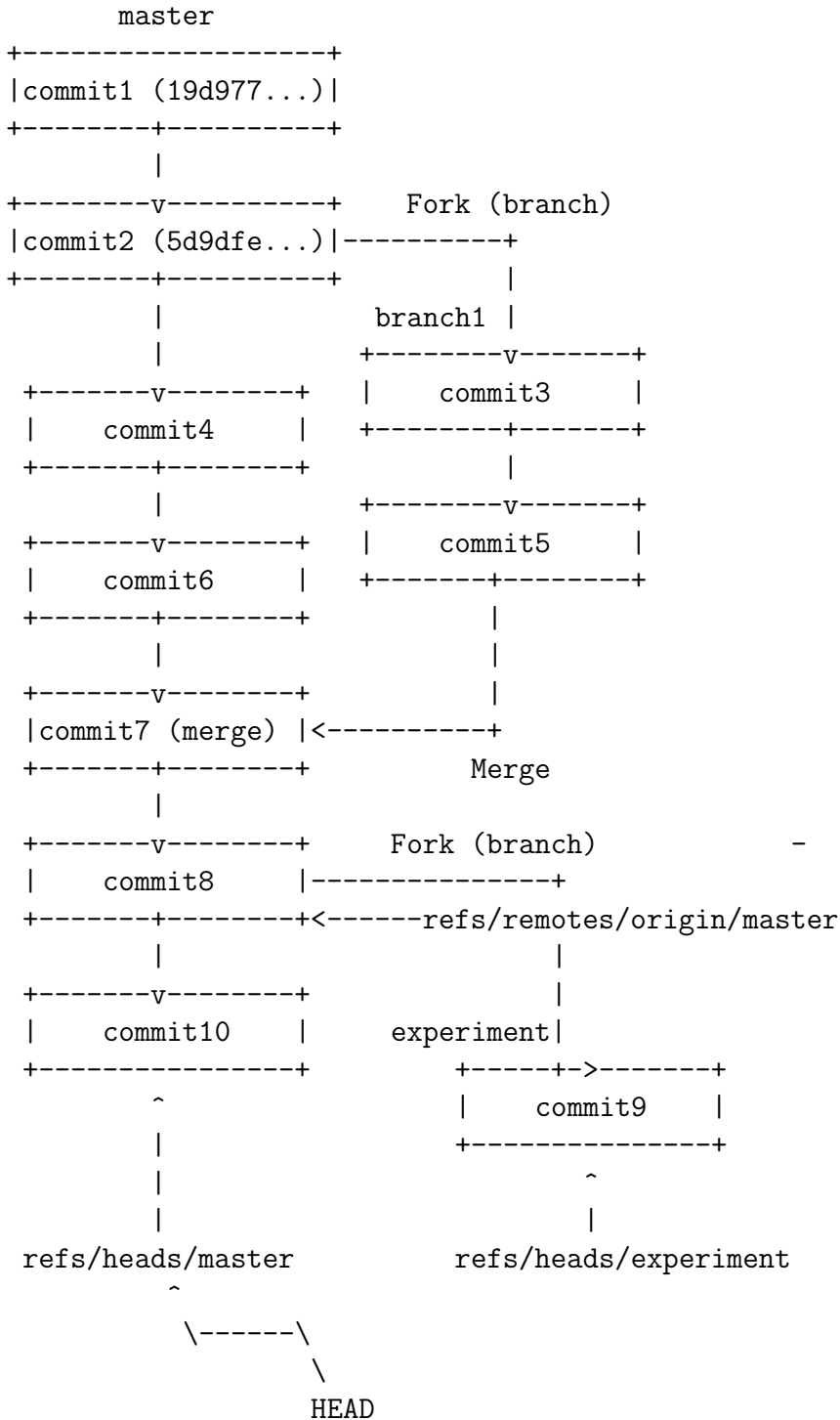


Figure 2.2: Commit graph and references.

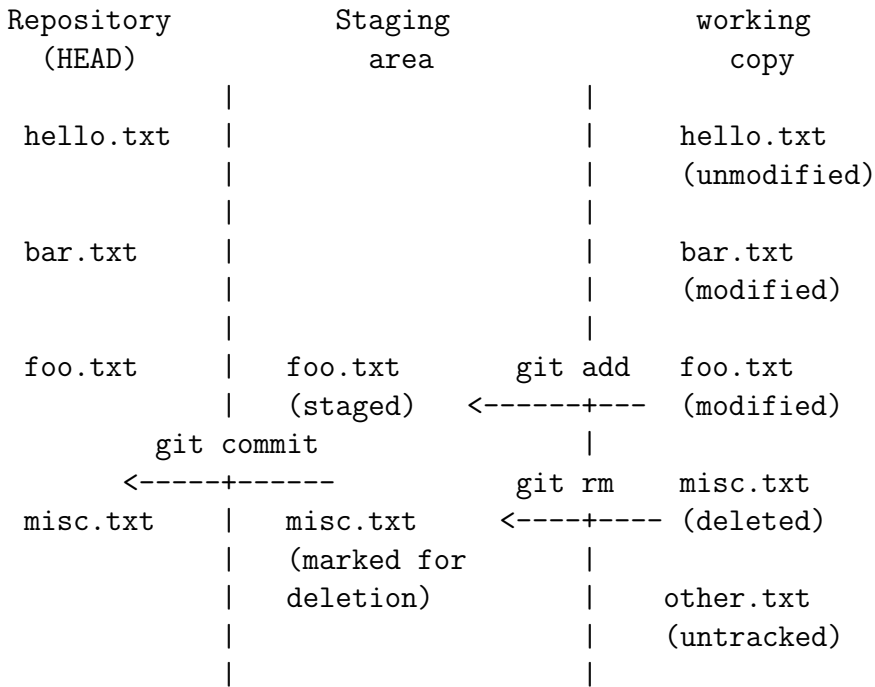


Figure 2.3: Git areas.

to the user by allowing to split a set of modifications in multiple commits. For example, in Figure 2.3, the modification to `bar.txt` can be put in another commit.

Internally, Git uses a data structure called the *index* (also known as the *directory cache*) to manage the staging area. I will explain later in Section 3.4 why this data structure is called an index.

2.4 Repository format: `.git/`

Now that you know the main concepts and data structures underlying Git, it will be easy to understand the format of the `.git/` directory. Here is the slightly edited output of the UNIX command `tree`, which recursively explores a directory and displays its content as a tree³. The command is applied here to the `/tests/hello/.git/` directory after the last command in Section 1.4, when the repository contained just one commit:

```

1 $ pwd
/tests/hello
2 $ tree -F .git/
.git/
|-- HEAD
...
|-- index
|-- objects/
|   |-- 2f/
|   |   '-- 092e9cadfc1eb4a6d2febfddb941f4c1fe6fd6
|   |-- 19/
|   |   '-- d977a48d1d7b7ae8d520dd66190702dd4fb5bc
|   |-- 9f/

```

³See <http://mama.indstate.edu/users/ice/tree/> for more information about the `tree` program. The `-F` option used in the terminal session above is to add the special mark `'/'` at the end of directory names, to help differentiate regular files from subdirectories.

```

|   |   '-- 4d96d5b00d98959ea9960f069585ce42b1349a
.....
'-- refs/
    |-- heads/
    |   '-- master
.....

```

The objects of the key/value store are simply stored in separate files under `.git/objects/`. The hexadecimal digits of the SHA1 of the object are split in two parts: the first two digits are used for the directory name containing the object and the remaining digits for the filename of the object. For example, the file `.git/objects/9f/4d96d5b00d98959ea9960f069585ce42b1349a` contains the blob of `Hello Git\n` (see the blob and its SHA1 at the top right of Figure 2.1).

Git uses the first two digits of the SHA1 to classify objects in separate folders to avoid having too many files in the same directory. Indeed, this would slow down filesystem operations such as opening a file referenced by a specific SHA1 key, a recurring operation under Git (see the `KERNEL` book [Pad14] for more information on the performance of filesystem operations).

The references are also stored simply in separate files. Git uses the name of the reference as the path to a file under `.git/refs/` (e.g., `.git/refs/heads/master` in the example above).

Finally, the index and the HEAD are stored directly under `.git/` in separate files.

2.5 hello/.git/

In this section, I will give a short tutorial on how to use Git. I will explain the main commands of `git` by using a terminal session starting from an empty `hello/` directory. I will continue the series of commands I introduced in Section 1.4. I will also describe the semantics of those commands by showing their effects on the Git metadata stored under the `hello/.git/` subdirectory (by using mainly the output of the `tree` command, as in the previous section). This will hopefully help you to understand the code of those commands I will present later in the following chapters.

I will again use `git` instead of `git9` in the terminal sessions but the examples are easy to transpose to Plan 9.

2.5.1 Creating a repository

Here are the commands to create a fresh new repository:

```

1 $ cd /tests/
2 $ mkdir hello
3 $ cd hello/
4 $ git init
Initialized empty Git repository in /tests/hello/.git/
5 $ tree -F .git/
.git/
|-- HEAD
...
|-- objects/
'-- refs/
    |-- heads/
6 $ cat .git/HEAD
ref: refs/heads/master

```

The `init` command creates just the directory structure under `.git/`, without any objects or references in it (the `.git/objects/` and `.git/refs/heads` are empty directories above). There is also not yet an index file. There is a `HEAD` file, but its contents, shown with command 6 above references an head that does not exist yet under `.git/refs/heads/`.

2.5.2 Staging a diff

Here is the command to add a new file (or a new version of a file) to the repository:

```
1 $ echo "Hello Git" > hello.txt
2 $ git add hello.txt
3 $ tree -F .git/
.git/
|-- HEAD
...
|-- index
...
|-- objects/
|   |-- 9f/
|   |   '-- 4d96d5b00d98959ea9960f069585ce42b1349a
...
'-- refs/
    |-- heads/
4 $ cat .git/objects/9f/4d96d5b00d98959ea9960f069585ce42b1349a
x^K\312\3110R04'\360H\315\311\311Wp\317,\341^B^@4\201^Ec
```

Adding the first file in a repository has two effects:

1. The creation of a new *blob object* (here in `.git/objects/9f/4d96...`) containing essentially the compressed form of the content of the added file (hence the cryptic output of the `cat` command above). I will fully explain in Chapter 4 the format of a blob on the disk.
2. The creation of the index file. Again, I will describe precisely the format of the index file in Chapter 4, but the index essentially associates to every tracked filenames by the repository the SHA1 of the blob *currently* corresponding to the filename. In the example above, the index will associate to the filename `hello.txt` the SHA1 `9f4d96...`

You must use the command `add` to add a new file to the repository, when the file was not yet tracked by the repository. However, you must also use the command `add` to stage the modifications on a tracked file as in Figure 2.3. In that case, the `add` command also creates a new blob with the current content of the file. It also updates the index so the filename of the added file now points to the SHA1 of the new blob.

Note that using `git add` on an unmodified file will have no effect. Indeed, Git will compute the SHA1 of the current content, which will be identical to the SHA1 of an existing blob. Updating the index will have also no effect because the filename will reference the same SHA1.

2.5.3 Committing a diff

Here is the command to commit to the repository what was previously staged:

```

1 $ git commit -m "first commit" --author "pad <todo@todo>"
2 $ tree -F .git/
.git/
|-- HEAD
|-- index
|-- objects/
|   |-- 2f/
|   |   '-- 092e9cadfc1eb4a6d2febfddb941f4c1fe6fd6
|   |-- 19/
|   |   '-- d977a48d1d7b7ae8d520dd66190702dd4fb5bc
|   |-- 9f/
|   |   '-- 4d96d5b00d98959ea9960f069585ce42b1349a
'-- refs/
    |-- heads/
    |   '-- master
3 $ cat .git/refs/heads/master
19d977a48d1d7b7ae8d520dd66190702dd4fb5bc

```

The `commit` command has three effects:

1. The creation of a new *tree object*, in the example above `.git/objects/2f/092e...` (see Figure 2.1 for the content of this tree). This tree derives from the content of the index. Because the repository contains only one directory (the toplevel root containing just `hello.txt`), `git commit` will create just one tree object, but in general when a project has many directories `git commit` may create many new tree objects.
2. The creation of a new *commit object*, here `.git/objects/19/d977...`, referencing the newly created tree object (again, see Figure 2.1 for the content of this commit and its relationship to the tree object).
3. The update of the content of the head of the current branch (`refs/heads/master` according to the content of `.git/HEAD`) to contain the SHA1 of the newly created commit object (here `19d977...`), as shown by command 3 above.

Note that it can be tedious to specify each time on the command-line your name and email through the `--author` flag of `git commit`. Git can also leverage a configuration file containing such information in a `.gitconfig` file in your home directory as explained in Section 6.2.

2.5.4 Managing branches

Here is the command to create a new branch:

```

1 $ git branch experiment
2 $ tree -F .git/
.git/
|-- HEAD
...
'-- refs/
    |-- heads/
    |   |-- experiment
    |   '-- master
...
3 $ cat .git/HEAD

```

```
ref: refs/heads/master
4 $ cat .git/refs/heads/master
19d977a48d1d7b7ae8d520dd66190702dd4fb5bc
5 $ cat .git/refs/heads/experiment
19d977a48d1d7b7ae8d520dd66190702dd4fb5bc
```

As you can see, creating a new branch is an extremely cheap operation under Git. It consists just in adding a file under `.git/refs/heads/` with the name of the new branch (here `.git/refs/heads/experiment`), and the SHA1 commit of the current branch for its content (here `19d977...`). However, creating a branch does not switch to this branch as shown by command 3 above.

Here is the command to switch to the new branch:

```
6 $ git checkout experiment
Switched to branch 'experiment'
7 $ cat .git/HEAD
ref: refs/heads/experiment
```

Switching to a branch is usually a more costly operation. `git checkout` modifies `HEAD` (as shown by command 7 above), which is fast, but it also recomputes the index from the tree object that is referenced in the commit of the head of the new branch, as well as from all its subtrees (see Section 14.5.1 for the full description of this operation). It also updates all the files so that they contain the content of the blobs referenced in the index. The more the new branch differs from the current branch, the more costly switching to this new branch will be.

With `git9` you will need to use `git/branch -n experiment` to create a new branch and it will also automatically switch to this branch. There is no `git/checkout`; everything is done with `git/branch`.

2.5.5 Inspecting objects

Git provides a few commands to query the repository. Those commands are useful to understand the internal structures of Git. They are also useful for doing some archeology on the history of a project. The first of those commands, `show`, shows the content of a Git object. `git show` takes the SHA1 of an object as a parameter (in hexadecimal format) and pretty prints its content in a readable format. Here are a few examples of this command:

```
1 $ tree -F .git/
...
|-- objects/
|   |-- 2f/
|   |   '-- 092e9cadfc1eb4a6d2febfddb941f4c1fe6fd6
|   |-- 19/
|   |   '-- d977a48d1d7b7ae8d520dd66190702dd4fb5bc
|   |-- 9f/
|   |   '-- 4d96d5b00d98959ea9960f069585ce42b1349a
...
2 $ git show 9f4d96d5b00d98959ea9960f069585ce42b1349a
Hello Git
3 $ git show 2f092e9cadfc1eb4a6d2febfddb941f4c1fe6fd6
tree 2f092e9cadfc1eb4a6d2febfddb941f4c1fe6fd6

hello.txt
```

```
4 $ git show 19d977a48d1d7b7ae8d520dd66190702dd4fb5bc
commit 19d977a48d1d7b7ae8d520dd66190702dd4fb5bc
Author: pad <todo@todo>
Date:   Fri Sep 29 14:04:46 2017 -0700
```

```
    first commit
```

```
diff --git a/hello.txt b/hello.txt
new file mode 100644
index 0000000..9f4d96d
--- /dev/null
+++ b/hello.txt
@@ -0,0 +1 @@
+Hello Git
```

The command `show` mainly opens the appropriate file under `.git/objects/`, decompresses the file, and finally displays its content.

Another important query command is `log` to see the full history of a repository. Here is the output of this command on the repository we used this far:

```
1 $ git log
commit 19d977a48d1d7b7ae8d520dd66190702dd4fb5bc
Author: pad <todo@todo>
Date:   Fri Sep 29 14:04:46 2017 -0700
```

```
    first commit
```

The command `log` mainly opens `.git/HEAD` to get a reference to the last commit. Then, it goes through the parent links of the commit object to recursively explore all the linked commit objects (stored again under `.git/objects/`).

2.5.6 Other commands

There are a few other important Git commands: `git merge`, `git status`, `git diff`, `git clone`, `git pull`, or `git push`, but I will introduce them later in this document.

2.6 Code organization

2.7 Software architecture

2.8 Book structure

Part I
Foundations

Chapter 3

Core Data Structures

Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious.

Fred Brooks

3.1 Secure Hash Algorithm (SHA1) hashes

3.1.1 Binary hashes

```
<struct Hash 30a>≡ (270)
struct Hash {
    byte h[20];
};
```

```
<global Zhash 30b>≡ (284b)
Hash    Zhash;
```

```
<function hasheq 30c>≡ (284b)
int
hasheq(Hash *a, Hash *b)
{
    return memcmp(a->h, b->h, sizeof(a->h)) == 0;
}
```

```
<function hashcmp 30d>≡ (277)
int
hashcmp(uchar *a, uchar *b, uint nbit)
{
    int x, y, i, r;

    i = nbit/8;
    r = memcmp(a, b, i);
    if(r != 0 || nbit % 8 == 0)
        return r;
    x = a[i] >> (8 - nbit % 8);
    y = b[i] >> (8 - nbit % 8);
    return x - y;
}
```

Uses GETBE32 and Hashsz 161c.

3.1.2 Hexadecimal hashes

<pragmas varargck 31a>≡ (270) 252a▷
#pragma varargck type "H" Hash

<gitinit() fmtinstall calls 31b>≡ (61f) 252b▷
fmtinstall('H', Hfmt);

Uses Hfmt() 31c.

<function Hfmt 31c>≡ (284b)
int
Hfmt(Fmt *fmt)
{
 Hash h;
 int i, n, l;
 char c0, c1;

 l = 0;
 h = va_arg(fmt->args, Hash);
 for(i = 0; i < sizeof h.h; i++){
 n = (h.h[i] >> 4) & 0xf;
 c0 = (n >= 10) ? n-10 + 'a' : n + '0';
 n = h.h[i] & 0xf;
 c1 = (n >= 10) ? n-10 + 'a' : n + '0';
 l += fprintf(fmt, "%c%c", c0, c1);
 }
 return l;
}

3.1.3 Conversion functions

<function hparse 31d>≡ (284b)
errorneg1
hparse(Hash *h/*OUT*/, char *b)
{
 int i, c0, c1;

 for(i = 0; i < nelem(h->h); i++){
 if((c0 = charval(b[2*i+0])) == ERROR_NEG1)
 return ERROR_NEG1;
 if((c1 = charval(b[2*i+1])) == ERROR_NEG1)
 return ERROR_NEG1;
 h->h[i] = (c0 << 4) | c1;
 }
 return OK_0;
}

Uses charval().

<function charval 31e>≡ (284b)
int
charval(int c)
{
 if(c >= '0' && c <= '9')
 return c - '0';
 if(c >= 'a' && c <= 'f')
 return c - 'a' + 10;
 if(c >= 'A' && c <= 'F')
 return c - 'A' + 10;
 // else

```

    werrstr("invalid hex char");
    return ERROR_NEG1;
}

```

```

⟨macro GETBE32 32a⟩≡ (270)
#define GETBE32(b)\
    (((b)[0] & 0xFFu1) << 24) | \
    (((b)[1] & 0xFFu1) << 16) | \
    (((b)[2] & 0xFFu1) << 8) | \
    (((b)[3] & 0xFFu1) << 0)

```

3.2 Object

```

⟨struct Object 32b⟩≡ (270)
struct Object {
    /* Git data */
    Hash    hash;
    // enum<Gxxx>
    int type;

    ⟨Object other fields 33d⟩
    ⟨Object cache fields 33b⟩
    ⟨Object indexing fields 163a⟩
    ⟨Object extra fields 249c⟩

    /* Everything below here gets cleared */

    // raw object content after decompression
    // option<ref_own<string>>, set after Cloaded
    char    *all;
    // ref_shared<string> point in Object.all after object header (used for GBlob too)
    char    *data;
    /* size excludes header */
    vlong   size;

    // those fields are set after Cparsed
    /* Significant win on memory use */
    union {
        ⟨Object union fields 34d⟩
    };
};

```

```

⟨function objcmp 32c⟩≡ (277)
int
objcmp(void *pa, void *pb)
{
    Object *a, *b;

    a = *(Object**)pa;
    b = *(Object**)pb;
    return memcmp(a->hash.h, b->hash.h, sizeof(a->hash.h));
}

```

Uses `cache()` 34a and `objectcrc()` 158c.

3.2.1 Object types: Gxxx

```
<enum Gxxx 33a>≡ (270)
enum Gxxx {
    GNone    = 0,

    GCommit = 1,
    GTree   = 2,
    GBlob   = 3,

    <Gxxx other cases 177b>
};
```

3.2.2 Object flags: Cxxx

```
<Object cache fields 33b>≡ (32b) 248e▷
// enum<Cxxx>
int flag;

<enum Cxxx 33c>≡ (270)
enum Cxxx {
    Cloaded = 1 << 0,
    Cparsed = 1 << 2,
    <Cxxx other cases 180a>
};
```

3.2.3 Reference counting

```
<Object other fields 33d>≡ (32b)
// shared objects
int refs;
```

```
<function ref 33e>≡ (277)
Object*
ref(Object *o)
{
    o->refs++;
    return o;
}
```

Uses Ccache 33c and lruhead 152a.

```
<function unref 33f>≡ (277)
void
unref(Object *o)
{
    if(!o)
        return;
    o->refs--;
    if(o->refs == 0)
        clear(o);
}
```

Uses lrutail 152a.

```

⟨function clear 34a⟩≡ (277)
  /// ?? -> <>
  static void
  clear(Object *o)
  {
    if(!o)
      return;

    assert(o->refs == 0);
    ⟨clear() assert o->flag 34b⟩

    switch(o->type){
    ⟨clear() switch o->type cases 35a⟩
    default:
      break;
    }

    free(o->all);
    o->all = nil;
    o->data = nil;
    ⟨clear() reset o->flag 34c⟩
  }

```

Uses Cloaded, Cparsed, lruhead 152a, objcache 152a, and osadd() 37e.

```

⟨clear() assert o->flag 34b⟩≡ (34a) 248d▷
  assert(o->flag & Cloaded);

```

```

⟨clear() reset o->flag 34c⟩≡ (34a)
  o->flag &= ~(Cloaded|Cparsed);

```

Uses lrutail 152a.

3.2.4 Blobs

3.2.5 Trees and Tinfo

```

⟨Object union fields 34d⟩≡ (32b) 36a▷
  // when GTree
  Tinfo *tree;

```

```

⟨struct Tinfo 34e⟩≡ (270)
  struct Tinfo {
    /* Tree */
    // array<ref_own<Dirent>> (len = nent)
    Dirent *ent;
    int nent;
  };

```

```

⟨struct Dirent 34f⟩≡ (270)
  struct Dirent {
    // ref_own<string>
    char *name;
    int mode;
    // hash of GBlob (file) or GTree (dir) but not GCommit
    Hash h;
    ⟨Dirent other fields 236a⟩
  };

```

`<clear() switch o->type cases 35a>≡`

`(34a) 36c>`

```
case GTree:
    if(!o->tree)
        break;
    free(o->tree->ent);
    free(o->tree);
    o->tree = nil;
    break;
```

Uses `clear()` 160c.

`<function entcmp 35b>≡`

`(284b)`

```
int
entcmp(void *pa, void *pb)
{
    Dirent *ae, *be;
    uchar *a, *b;
    int ca, cb;

    ae = pa;
    be = pb;
    a = (uchar*)ae->name;
    b = (uchar*)be->name;
    /*
     * If the files have the same name, they're equal.
     * Otherwise, If they're trees, they sort as though
     * there was a trailing slash.
     *
     * Wat.
     */
    while(true){
        ca = *a++;
        cb = *b++;
        /*
         * because these are dir entries in a tree,
         * the only '/' allowable is the virtual '/'
         * at the end of the file name.
         */
        assert(ca != '/' && cb != '/');
        if(ca != cb){
            if(ca == 0 && (ae->mode & DMDIR))
                ca = '/';
            if(cb == 0 && (be->mode & DMDIR))
                cb = '/';
            return (ca > cb) ? 1 : -1;
        }
        if(ca == 0){
            if(ae->mode & DMDIR)
                ca = '/';
            if(be->mode & DMDIR)
                cb = '/';
            if(ca == cb)
                return 0;
            return (ca > cb) ? 1 : -1;
        }
    }
}
```

`<function emptydir 35c>≡`

`(284b)`

```
Object*
emptydir(void)
```

```

{
    static Object *e;

    if(e != nil)
        return ref(e);
    // else
    e = emalloc(sizeof(Object));
    e->type = GTree;
    e->tree = emalloc(sizeof(Tinfo));
    e->tree->ent = nil;
    e->tree->nent = 0;
    e->flag |= Cloaded|Cparsed;
    e->hash = Zhash;
    e->off = -1;
    ref(e);
    cache(e);

    return e;
}

```

Uses Cloaded, Cparsed, GTree 33a, Zhash 30b, cache() 34a, and ref() 36c.

3.2.6 Commits and Cinfo

```

<Object union fields 36a>+≡ (32b) <34d
    // when GCommit
    Cinfo *commit;

```

```

<struct Cinfo 36b>≡ (270)
struct Cinfo {
    /* Commit */
    // array<Hash> (0, 1, or 2 parents?)
    Hash *parent;
    int nparent;

    Hash tree;

    // ref_own<string>
    char *author;
    // option<ref_own<string>>
    char *committer;

    // ref<string> (len = nmsg), point in Object.all?
    char *msg;
    int nmsg;

    vlong ctime;
    vlong mtime;
};

```

```

<clear() switch o->type cases 36c>+≡ (34a) <35a
case GCommit:
    if(!o->commit)
        break;
    free(o->commit->parent);
    free(o->commit->author);
    free(o->commit->committer);
    free(o->commit);
    o->commit = nil;
    break;

```

```

⟨global zcommit(ref.c 37a)≡ (281)
    static Object zcommit = {
        .type=GCommit
    };

```

3.3 Objects

3.3.1 Object set: Objset

```

⟨struct Objset 37b)≡ (270)
    struct Objset {
        // hash_array<Hash, ref<Object>> (used = nobj, allocated = sz)
        Object **obj;
        int nobj;
        int sz;
    };

```

```

⟨function osinit 37c)≡ (273a)
    void
    osinit(Objset *s)
    {
        s->sz = 16;
        s->nobj = 0;
        s->obj = eamalloc(s->sz, sizeof(Object*));
    }

```

Uses eamalloc().

```

⟨function osclear 37d)≡ (273a)
    void
    osclear(Objset *s)
    {
        free(s->obj);
    }

```

Objset methods

```

⟨function osadd 37e)≡ (273a)
    void
    osadd(Objset *s, Object *o)
    {
        u32int probe;
        ⟨osadd() other locals 38a)

        probe = GETBE32(o->hash.h) % s->sz;
        while(s->obj[probe]){
            if(hasheq(&s->obj[probe]->hash, &o->hash)){
                // update the object (old = leak?)
                s->obj[probe] = o;
                return;
            }
            // else
            probe = (probe + 1) % s->sz;
        }
        assert(s->obj[probe] == nil);
        s->obj[probe] = o;
        s->nobj++;
    }

```

```

    ⟨osadd() realloc if needed 38b⟩
}

```

Uses GETBE32 and hasheq().

```

⟨osadd() other locals 38a⟩≡ (37e)
Object **obj;
int i, sz;

```

```

⟨osadd() realloc if needed 38b⟩≡ (37e)
if(s->sz < 2*s->nobj){
    sz = s->sz;
    obj = s->obj;

    s->sz *= 2;
    s->nobj = 0;
    s->obj = eamalloc(s->sz, sizeof(Object*));
    for(i = 0; i < sz; i++)
        if(obj[i])
            // recurse! need to maintain hash invariant
            osadd(s, obj[i]);
    free(obj);
}

```

Uses eamalloc() and osadd() 37e.

```

⟨function osfind 38c⟩≡ (273a)
Object*
osfind(Objset *s, Hash h)
{
    u32int probe;

    for(probe = GETBE32(h.h) % s->sz; s->obj[probe]; probe = (probe + 1) % s->sz)
        if(hasheq(&s->obj[probe]->hash, &h))
            return s->obj[probe];
    // else
    return nil;
}

```

Uses GETBE32 and hasheq().

```

⟨function oshas 38d⟩≡ (273a)
int
oshas(Objset *s, Hash h)
{
    return osfind(s, h) != nil;
}

```

3.3.2 Object queue: Objq

```

⟨struct Objq 38e⟩≡ (270)
struct Objq {
    // growing_array<Qelt> (used = nheap, allocated = heapsz)
    Qelt *heap;
    int nheap;
    int heapsz;
};

```

```

<struct Qelt 39a>≡ (270)
struct Qelt {
    // ref<Object>
    Object *o;

    vlong ctime;

    //enum<Qcolor>
    int color;
};

```

Objq methods

```

<function qinit 39b>≡ (284b)
void
qinit(Objq *q)
{
    memset(q, 0, sizeof(Objq));
    q->nheap = 0;
    q->heapsz = 8;
    q->heap = eamalloc(q->heapsz, sizeof(Qelt));
}

```

```

<function qclear 39c>≡ (284b)
void
qclear(Objq *q)
{
    free(q->heap);
}

```

```

<function qput 39d>≡ (284b)
void
qput(Objq *q, Object *o, int color)
{
    Qelt t;
    int i;

    assert(o->type == GCommit);
    <qput() realloc if needed 39e>
    q->heap[q->nheap].o = o;
    q->heap[q->nheap].color = color;
    q->heap[q->nheap].ctime = o->commit->ctime;
    <qput() resort by ctime 39f>
    q->nheap++;
}

```

Uses interactive.

```

<qput() realloc if needed 39e>≡ (39d)
if(q->nheap == q->heapsz){
    q->heapsz *= 2;
    q->heap = earealloc(q->heap, q->heapsz, sizeof(Qelt));
}

```

```

<qput() resort by ctime 39f>≡ (39d)
for(i = q->nheap; i > 0; i = (i-1)/2){
    if(q->heap[i].ctime < q->heap[(i-1)/2].ctime)
        break;
    // else exchange
    t = q->heap[i];
}

```

```

    q->heap[i] = q->heap[(i-1)/2];
    q->heap[(i-1)/2] = t;
}

```

Uses `eamalloc()`.

`<function qpop 40a>≡ (284b)`

```

int
qpop(Objq *q, Qelt *e)
{
    int i, l, r, m;
    Qelt t;

    if(q->nheap == 0)
        return 0;
    *e = q->heap[0];
    if(--q->nheap == 0)
        return 1;

    i = 0;
    q->heap[0] = q->heap[q->nheap];
    while(true){
        m = i;
        l = 2*i+1;
        r = 2*i+2;
        if(l < q->nheap && q->heap[m].ctime < q->heap[l].ctime)
            m = l;
        if(r < q->nheap && q->heap[m].ctime < q->heap[r].ctime)
            m = r;
        if(m == i)
            break;
        t = q->heap[m];
        q->heap[m] = q->heap[i];
        q->heap[i] = t;
        i = m;
    }
    return 1;
}

```

Uses `GCommit 33a` and `earealloc() 257a`.

3.4 .git/INDEX9

3.4.1 Status character: RMAUT

`<enum WalkFlags 40b>≡ (285)`

```

// do not reorder, some code assume Tflg is the last
enum WalkFlags {
    Rflg    = 1 << 0,
    Mflg    = 1 << 1,
    Aflg    = 1 << 2,
    Uflg    = 1 << 3,
    /* everything after this is not an error */
    Tflg    = 1 << 4,
};

```

`<global printflg(walk.c) 40c>≡ (285)`

```

// bitset<enum<WalkFlags>>
int printflg;

```

3.4.2 Idxent

```
<struct Idxent 41a>≡ (270)
struct Idxent {
    // ref_own<string>
    char    *path;
    // option<Qid>? (None = 0)
    Qid qid;
    int mode;
    // global counter (used in idxcmp())
    int order;
    // RMAUT?
    char    state;
};
```

```
<function idxcmp 41b>≡ (282b)
int
idxcmp(void *pa, void *pb)
{
    Idxent *a, *b;
    int c;

    a = (Idxent*)pa;
    b = (Idxent*)pb;
    if((c = strcmp(a->path, b->path)) != 0)
        return c;
    assert(a->order != b->order);
    return a->order < b->order ? -1 : 1;
}
```

```
<function idxcmp (git9/walk.c) 41c>≡ (285)
int
idxcmp(void *pa, void *pb)
{
    Idxent *a, *b;
    int c;

    a = (Idxent*)pa;
    b = (Idxent*)pb;
    if((c = strcmp(a->path, b->path)) != 0)
        return c;
    /* maintain load order if name is identical */
    return a->order < b->order ? -1 : 1;
}
```

3.4.3 idx

```
<globals idx(save.c) 41d>≡ (282b)
// growing_array<Idxent> (used = nidx, allocated = idxsz)
Idxent *idx;
int idxsz;
int nidx;
```

```
<main()(save.c) initialize idx 41e>≡ (53b)
nidx = 0;
idxsz = 32;
idx = emalloc(idxsz*sizeof(Idxent));
```

```
<main() (save.c) realloc idx if needed 42a>≡ (53b)
    if(nidx == idxsz){
        idxsz += idxsz/2;
        idx = realloc(idx, idxsz*sizeof(Idxent));
    }
```

```
<globals idx(walk.c) 42b>≡ (285)
    // growing_array<Idxent> (used = nidx, allocated = idxsz)
    Idxent *idx;
    int idxsz;
    int nidx;
```

```
<main() (walk.c) initialize idx 42c>≡ (52c)
    nidx = 0;
    idxsz = 32;
    idx = emalloc(idxsz*sizeof(Idxent)); // why not eamalloc?
```

```
<main() (walk.c) realloc idx if needed 42d>≡ (52c)
    if(nidx == idxsz){
        idxsz += idxsz/2;
        idx = erealloc(idx, idxsz*sizeof(Idxent));
    }
```

Chapter 4

Reading from a Repository

4.1 Objects

4.1.1 readobject()

```
<function readobject 43a>≡ (277)
  /// gitwalk1 | gtreegen | ... -> <>
  /*
   * Loads and returns a cached object.
  */
  Object*
  readobject(Hash h)
  {
    Object *o;

    o = readidxobject(nil, h, 0);
    if(o == nil)
      return nil;
    // else
    parseobject(o);
    ref(o);
    return o;
  }
```

4.1.2 readidxobject() part 1

```
<function readidxobject 43b>≡ (277)
  /// readobject | readrdelta -> <>
  static Object*
  readidxobject(Biobuf *idx, Hash h, int flag)
  {
    char path[Pathmax], hbuf[41];
    Object *obj, *new;
    int i, r;
    bool retried;
    Biobuf *f;
    vlong o;

    <readidxobject() if h in object cache 248f>
    <readidxobject() if Cthin flag 180b>
    <readidxobject() if Cidx flag 180d>
    // else
    new = nil;
    if(obj == nil){
```

```

    new = emalloc(sizeof(Object));
    new->id = objcache.nobj + 1;
    new->hash = h;
    obj = new;
}

o = -1;
retried = false;
retry:
    <readidobject() look for object in packs 157b>
    // else
    snprintf(hbuf, sizeof(hbuf), "%H", h);
    snprintf(path, sizeof(path), ".git/objects/%c%c/%s", hbuf[0], hbuf[1], hbuf + 2);
    if((f = Bopen(path, OREAD)) != nil){
        if(readloose(f, obj, flag) == ERROR_NEG1)
            goto errorf;
        Bterm(f);
        parseobject(obj);
        cache(obj);
        return obj;
    }
    // else

    if(o == -1){
        if(retried)
            goto error;
        retried = true;
        <readidobject() when object not found call refreshpacks() 158a>
        goto retry;
    }
errorf:
    Bterm(f);
error:
    free(new);
    return nil;
}

```

Uses objcache 152a, osadd() 37e, osfind() 38c, and refreshpacks().

4.1.3 readloose() and Cloaded

```

<function readloose 44>≡ (277)
    /// readobject -> readidobject | ??? -> <>
    static errorneg1
    readloose(Biobuf *f, Object *o, int flag)
    {
        struct { char *tag; int type; } *p, types[] = {
            {"blob", GBlob},
            {"tree", GTree},
            {"commit", GCommit},
            <readloose() local types other elements 238d>
            {nil},
        };
        char *d, *s, *e, *ed;
        vlong sz, n;
        int l;

        n = decompress(&d, f, nil);
        <readloose() sanity check n 45a>
    }

```

```

s = d;
ed = d + n;
o->type = GNone;
for(p = types; p->tag; p++){
    l = strlen(p->tag);
    if(strncmp(s, p->tag, l) == 0){
        s += l;
        o->type = p->type;
        while(s != ed && !isspace(*s))
            s++;
        break;
    }
}
if(o->type == GNone){
    free(o->data);
    return ERROR_NEG1;
}
// else
sz = strtol(s, &e, 0);
<readloose() sanity check e 45b>
<readloose() sanity check sz 45c>
o->all = d;
o->data = e;
o->size = sz;
o->flag |= Cloaded|flag;
return OK_0;

```

```

error:
    free(d);
    return ERROR_NEG1;
}

```

Uses GETBE32.

```

<readloose() sanity check n 45a>≡ (44)
if(n == ERROR_NEG1)
    return ERROR_NEG1;

```

```

<readloose() sanity check e 45b>≡ (44)
if(e == s || *e++ != '\0'){
    werrstr("malformed object header");
    goto error;
}

```

```

<readloose() sanity check sz 45c>≡ (44)
if(sz != n - (e - d)){
    werrstr("mismatched sizes");
    goto error;
}

```

4.1.4 parseobject() and Cparsed

```

<function parseobject 45d>≡ (277)
void
parseobject(Object *o)
{
    if(o->flag & Cparsed)
        return;
    switch(o->type){
        <parseobject() switch o->type cases 47b>

```

```

    default:    break;
}
o->flag |= Cparsed;
}

```

Uses `closepack()` 158d, `packf` 152c, `parseobject()` 48c, `readloose()` 161e, and `readpacked()` 180e.

4.1.5 Decompression

```

⟨gitinit() libflate initializations 46a⟩≡ (61f)
inflateinit();
deflateinit();

```

```

⟨function decompress 46b⟩≡ (277)
/// readloose | ??? -> <>
errorneg1
decompress(void **p, Biobuf *b, vlong *csz)
{
    Buf d = {.len=0, .data=nil, .sz=0};

    if(bdecompress(&d, b, csz) == ERROR_NEG1){
        free(d.data);
        return ERROR_NEG1;
    }
    *p = d.data;
    return d.len;
}

```

```

⟨struct Buf 46c⟩≡ (277)
struct Buf {
    int len;
    int sz;
    int off;
    char *data;
};

```

```

⟨function bdecompress 46d⟩≡ (277)
int
bdecompress(Buf *d, Biobuf *b, vlong *csz)
{
    vlong o;

    o = Boffset(b);
    if(inflatezlib(d, bappend, b, breadc) == -1 || d->data == nil){
        free(d->data);
        return -1;
    }
    if (csz)
        *csz = Boffset(b) - o;
    return d->len;
}

```

Uses `readvint()` 47a.

```

⟨function breadc 46e⟩≡ (277)
int
breadc(void *p)
{
    return Bgetc(p);
}

```

```

⟨function bappend 47a⟩≡ (277)
int
bappend(void *p, void *src, int len)
{
    Buf *b = p;
    char *n;

    while(b->len + len >= b->sz){
        b->sz = b->sz*2 + 64;
        n = realloc(b->data, b->sz);
        if(n == nil)
            return -1;
        b->data = n;
    }
    memmove(b->data + b->len, src, len);
    b->len += len;
    return len;
}

```

4.1.6 Reading a blob

4.1.7 Reading a tree

```

⟨parseobject() switch o->type cases 47b⟩≡ (45d) 48b▷
    case GTree: parsetree(o);    break;

```

```

⟨function parsetree 47c⟩≡ (277)
static void
parsetree(Object *o)
{
    int m, a, entsz, nent;
    Dirent *t, *ent;
    char *p, *ep;

    p = o->data;
    ep = p + o->size;

    nent = 0;
    entsz = 16;
    ent = eamalloc(entsz, sizeof(Dirent));

    o->tree = emalloc(sizeof(Tinfo));
    while(p != ep){
        ⟨parsetree() realloc ent if needed 48a⟩
        t = &ent[nent++];
        m = strtol(p, &p, 8);
        if(*p != ' ')
            sysfatal("malformed tree %H: *p=(%d) %c", o->hash, *p, *p);
        p++;
        /*
         * only the stored permissions for the user
         * are relevant; git fills group and world
         * bits with whatever -- so to serve with
         * useful permissions, replicate the mode
         * of the git repo dir.
         */
        t->mode = gitdirmode;
        t->ismod = false;
        t->islink = false;
    }
}

```

```

if(m & 0777){
    a = (m & 0777)>>6;
    t->mode &= ((a<<6)|(a<<3)|a);
}
⟨parsetree() if submodule 237b⟩
⟨parsetree() if symlink 236b⟩
if(m & 0040000) /* dir */
    t->mode |= DMDIR;
t->name = p;
p = memchr(p, 0, ep - p);
if(p == nil || *p++ != 0 || ep - p < sizeof(t->h.h))
    sysfatal("malformed tree %H, remaining %d (%s)", o->hash, (int)(ep - p), p);
memcpy(t->h.h, p, sizeof(t->h.h));
p += sizeof(t->h.h);
}
o->tree->ent = ent;
o->tree->nent = nent;
}

```

Uses Cidx 33c, Cloaded, Cparsed, GTag 177b, Pathmax 270, objcache 152a, osfind() 38c, parsetag() 48c, and readpacked() 180e.

```

⟨parsetree() realloc ent if needed 48a⟩≡ (47c)
if(nent == entsz){
    entsz *= 2;
    ent = earealloc(ent, entsz, sizeof(Dirent));
}

```

4.1.8 Reading a commit

```

⟨parseobject() switch o->type cases 48b⟩+≡ (45d) <47b 238e>
case GCommit: parsecommit(o); break;

```

```

⟨function parsecommit 48c⟩≡ (277)
static void
parsecommit(Object *o)
{
    char *p, *t, buf[512];
    int np;

    p = o->data;
    np = o->size;
    o->commit = emalloc(sizeof(Cinfo));
    while(true){
        if(scanword(&p, &np, buf, sizeof(buf)) == -1)
            break;
        if(strcmp(buf, "tree") == 0){
            if(scanword(&p, &np, buf, sizeof(buf)) == -1)
                sysfatal("invalid commit: tree missing");
            if(hparse(&o->commit->tree, buf) == -1)
                sysfatal("invalid commit: garbled tree");
        }else if(strcmp(buf, "parent") == 0){
            if(scanword(&p, &np, buf, sizeof(buf)) == -1)
                sysfatal("invalid commit: missing parent");
            o->commit->parent = realloc(o->commit->parent, ++o->commit->nparent * sizeof(Hash));
            if(!o->commit->parent)
                sysfatal("unable to malloc: %r");
            if(hparse(&o->commit->parent[o->commit->nparent - 1], buf) == -1)
                sysfatal("invalid commit: garbled parent");
        }else if(strcmp(buf, "author") == 0){
            parseauthor(&p, &np, &o->commit->author, &o->commit->mtime);
        }
    }
}

```

```

}else if(strcmp(buf, "committer") == 0){
    parseauthor(&p, &np, &o->commit->committer, &o->commit->ctime);
}else if(strcmp(buf, "gpgsig") == 0){
    /* just drop it */
    if((t = strstr(p, "-----END PGP SIGNATURE-----")) == nil)
    if((t = strstr(p, "-----END SSH SIGNATURE-----")) == nil)
    if((t = strstr(p, "-----END SIGNED MESSAGE-----")) == nil)
        sysfatal("malformed gpg signature");
    np -= t - p;
    p = t;
}
nextline(&p, &np);
}
while (np && isspace(*p)) {
    p++;
    np--;
}
o->commit->msg = p;
o->commit->nmsg = np;
}

```

Uses Cparsed and gitdirmode.

parseauthor()

```

<gitinit() set authorpat 49a>≡ (61f)
    authorpat = regcomp("[\t ]*(.*)[\t ]+([0-9]+)[\t ]*([\-\+]?[0-9]+)?");

```

Uses authorpat 284b.

```

<function parseauthor 49b>≡ (277)

```

```

// may raise sysfatal
static void
parseauthor(char **str, int *nstr, char **name, vlong *time)
{
    char buf[512];
    Resub m[4];
    vlong tz;
    char *p;
    int n, nm;

    p = strchr(*str, '\n');
    <parseauthor() sanity check p 50a>
    n = p - *str;
    <parseauthor() sanity check n 50b>
    memset(m, 0, sizeof(m));
    memcpy(buf, *str, n);
    buf[n] = '\0';
    *str = p;
    *nstr -= n;

    if(!regexec(authorpat, buf, m, nelem(m)))
        sysfatal("invalid author line %s", buf);
    nm = m[1].e.ep - m[1].s.sp;
    *name = emalloc(nm + 1);
    memcpy(*name, m[1].s.sp, nm);
    buf[nm] = '\0';

    nm = m[3].e.ep - m[3].s.sp;
    memcpy(buf, m[3].s.sp, nm);
    buf[nm] = '\0';
}

```

```

tz = atoll(buf);

nm = m[2].e.ep - m[2].s.sp;
memcpy(buf, m[2].s.sp, nm);
buf[nm] = '\0';
*time = atoll(buf) + 3600*(tz/100) + 60*(tz%100);

return;
}

```

Uses `eamalloc()` and `earealloc()` [257a](#).

```

⟨parseauthor() sanity check p 50a⟩≡ (49b)
if(p == nil)
    sysfatal("malformed author line");

```

```

⟨parseauthor() sanity check n 50b⟩≡ (49b)
if(n >= sizeof(buf))
    sysfatal("overlong author line");

```

4.2 References

4.2.1 readref()

```

⟨function readref (git9/ref.c) 50c⟩≡ (281)
errorneg1
readref(Hash *h/*OUT*/, char *ref)
{
    char buf[256], s[256];
    errorneg1 r;
    fdt f;
    int n;
    ⟨readref() (ref.c) other locals 51c⟩

    r = hparse(h, ref);
    if(r != ERROR_NEG1)
        return r;
    // else
    if(strcmp(ref, "HEAD") == 0){
        snprintf(buf, sizeof(buf), ".git/HEAD");
        f = open(buf, OREAD);
        ⟨readref(ref.c) sanity check f 51a⟩
        n = readn(f, s, sizeof(s) - 1);
        ⟨readref(ref.c) sanity check n 51b⟩
        // else
        s[n] = '\0';
        strip(s);
        r = hparse(h, s);
        goto found;
    }
    // else
    ⟨readref() (ref.c) try .git/refs prefixes 51d⟩
    ⟨readref() (ref.c) try other prefixes 51e⟩
    // else
    return ERROR_NEG1;

found:
    if(r == ERROR_NEG1 && strncmp(s, "ref: ", 5) == 0)
        // recurse!

```

```

    r = readref(h, s + 5);
    return r;
}

```

Uses `hparse()` 31d.

```

⟨readref(ref.c) sanity check f 51a⟩≡ (50c)
    if(f == ERROR_NEG1)
        return ERROR_NEG1;

```

```

⟨readref(ref.c) sanity check n 51b⟩≡ (50c)
    if(n == ERROR_NEG1)
        return ERROR_NEG1;

```

4.2.2 .refs/ prefixes

```

⟨readref() (ref.c) other locals 51c⟩≡ (50c)
    static char *try[] =
        {"", "refs/", "refs/heads/", "refs/remotes/", "refs/tags/", nil};
    char **pfx;

```

```

⟨readref() (ref.c) try .git/refs prefixes 51d⟩≡ (50c)
    for(pfx = try; *pfx; pfx++){
        snprintf(buf, sizeof(buf), ".git/%s%s", *pfx, ref);
        f = open(buf, OREAD);
        if(f == ERROR_NEG1)
            continue;
        n = readn(f, s, sizeof(s) - 1);
        if(n == ERROR_NEG1)
            // not closing??
            continue;
        // else
        s[n] = '\0';
        strip(s);
        r = hparse(h, s);
        close(f);
        goto found;
    }

```

Uses `hparse()` 31d and `matchpfx()` 51f.

4.2.3 matchpfx()

```

⟨readref() (ref.c) try other prefixes 51e⟩≡ (50c)
    if((r = matchpfx(h, ref)) != ERROR_NEG1)
        return r;

```

```

⟨function matchpfx 51f⟩≡ (281)
    static errorneg1
    matchpfx(Hash *h, char *ref)
    {
        int i, c;
        Hash pfx;
        char *p;

        memset(&pfx, 0, sizeof(Hash));
        for(i = 0, p = ref; *p; p++, i++){
            if((c = charval(*p)) == -1)
                return ERROR_NEG1;
            pfx.h[i/2] |= c;
        }
    }

```

```

    if((i & 1) == 0)
        pfx.h[i/2] <<= 4;
}
return expandprefix(h, pfx, i*4);
}

```

Uses `charval()` and `expandprefix()` 248f.

<function expandprefix 52a>≡ (277)

```

errorneg1
expandprefix(Hash *rh/*OUT*/, Hash h, int npfx)
{
    int i, ndir;
    fdt fd;
    char buf[128];
    Dir *d;

    <expandprefix() call refreshpacks() 158b>

    if(npfx < 8 || npfx % 4 != 0)
        return ERROR_NEG1;
    <expandprefix() iterate on packf 158c>
    // else
    sprintf(buf, ".git/objects/%x", h.h[0]);
    if((fd = open(buf, OREAD)) == -1)
        return ERROR_NEG1;
    ndir = dirreadall(fd, &d);
    close(fd);
    if(ndir == -1)
        return ERROR_NEG1;
    for(i = 0; i < ndir; i++){
        sprintf(buf, sizeof(buf), "%x%s", h.h[0], d[i].name);
        if(hparse(rh, buf) == 0 && hashcmp(h.h, rh->h, npfx) == 0){
            free(d);
            return OK_0;
        }
    }

    free(d);
    return ERROR_NEG1;
}

```

Uses `crc32()`.

4.3 Index

<main() (walk.c) other locals 52b>≡ (102g) 105b▷

```

char *ln;
int line;
char *parts[4];

```

<main() (walk.c) initializations part 2, when has .git/INDEX9 52c>≡ (102g)

```

<main() (walk.c) initialize idx 42c>
line = 0;
while((ln = Brdstr(f, '\n', 1)) != nil){
    line++;
    /* allow blank lines */
    if(ln[0] == '\0' || ln[0] == '\n')
        continue;
    if(getfields(ln, parts, nelem(parts), 0, " \t") != nelem(parts))

```

```

        sysfatal(".git/INDEX9:%d: corrupt index", line);
// else
cleanname(parts[3]);
⟨main() (walk.c) realloc idx if needed 42d⟩
idx[nidx].state = *parts[0];
idx[nidx].qid = parseqid(parts[1]);
idx[nidx].mode = strtol(parts[2], nil, 8);
idx[nidx].path = estrdup(parts[3]);
idx[nidx].order = nidx;
nidx++;
free(ln);
}
Bterm(f);

⟨main() (save.c) other locals 53a⟩≡ (115e) 120c▷
char *ln;
int line;
char *parts[4];

⟨main() (save.c) read .git/INDEX9 and set idx 53b⟩≡ (115e)
⟨main() (save.c) initialize idx 41e⟩
line = 0;
while((ln = Brdstr(f, '\n', 1)) != nil){
    line++;
    if(ln[0] == '\0' || ln[0] == '\n')
        continue;
    if(getfields(ln, parts, nelem(parts), 0, " \t") != nelem(parts))
        sysfatal(".git/INDEX9:%d: corrupt index", line);
    cleanname(parts[3]);
    ⟨main() (save.c) realloc idx if needed 42a⟩
    idx[nidx].state = *parts[0];
    idx[nidx].qid = parseqid(parts[1]);
    idx[nidx].mode = strtol(parts[2], nil, 8);
    idx[nidx].path = strdup(parts[3]);
    idx[nidx].order = nidx;
    nidx++;
    free(ln);
}
Bterm(f);

⟨function parseqid 53c⟩≡ (284b)
Qid
parseqid(char *s)
{
    char *e;
    Qid q;

    if(strcmp(s, "NOQID") == 0)
        return (Qid){-1, -1, -1};
    e = s;
    q.path = strtoull(e, &e, 16);
    if(*e != '.')
        sysfatal("corrupt qid: %s (%s)", s, e);
    q.vers = strtoul(e+1, &e, 10);
    if(*e != '.')
        sysfatal("corrupt qid: %s (%s)", s, e);
    q.type = strtoul(e+1, &e, 16);
    if(*e != '\0')
        sysfatal("corrupt qid: %s (%x)", s, *e);
    return q;
}

```

}

Uses Seed-38 284b.

Chapter 5

Writing to a Repository

5.1 Objects

5.1.1 Objbuf

```
<struct Objbuf 55a>≡ (282b)
struct Objbuf {
    int off;
    char *hdr;
    int nhdr;
    char *dat;
    int ndat;
};
```

5.1.2 writeobj()

```
<function writeobj 55b>≡ (282b)
/// blobify | writetree | mkcommit -> <>
void
writeobj(Hash *h/*OUT*/, char *hdr, int nhdr, char *dat, int ndat)
{
    Objbuf b = {.off=0, .hdr=hdr, .nhdr=nhdr, .dat=dat, .ndat=ndat};
    char s[64], o[256];
    SHA1state *st;
    Biobuf *f;
    int fd;

    st = sha1((uchar*)hdr, nhdr, nil, nil);
    st = sha1((uchar*)dat, ndat, nil, st);
    sha1(nil, 0, h->h, st);

    snprintf(s, sizeof(s), "%H", *h);
    fd = create(".git/objects", OREAD, DMDIR|0755);
    close(fd);
    snprintf(o, sizeof(o), ".git/objects/%c%c", s[0], s[1]);
    fd = create(o, OREAD, DMDIR | 0755);
    close(fd);
    snprintf(o, sizeof(o), ".git/objects/%c%c/%s", s[0], s[1], s + 2);
    if(readobject(*h) == nil){
        f = Bopen(o, OWRITE);
        if(f == nil)
            sysfatal("could not open %s: %r", o);
        <writeobj() compress b in f 56a>
        Bterm(f);
    }
```

```

    }
}

```

Uses `gitmode()` 115c.

5.1.3 Compression

```

⟨writeobj() compress b in f 56a⟩≡ (55b)
    if(deflatezlib(f, bwrite, &b, objbytes, 9, 0) == ERROR_NEG1)
        sysfatal("could not write %s: %r", o);

```

```

⟨function bwrite 56b⟩≡ (282b)
    static int
    bwrite(void *p, void *buf, int nbuf)
    {
        return Bwrite(p, buf, nbuf);
    }

```

```

⟨function objbytes 56c⟩≡ (282b)
    static int
    objbytes(void *p, void *buf, int nbuf)
    {
        Objbuf *b;
        int r, n, o;
        char *s;

        b = p;
        n = 0;
        if(b->off < b->nhdr){
            r = b->nhdr - b->off;
            r = (nbuf < r) ? nbuf : r;
            memcpy(buf, b->hdr, r);
            b->off += r;
            nbuf -= r;
            n += r;
        }
        if(b->off < b->ndat + b->nhdr){
            s = buf;
            o = b->off - b->nhdr;
            r = b->ndat - o;
            r = (nbuf < r) ? nbuf : r;
            memcpy(s + n, b->dat + o, r);
            b->off += r;
            n += r;
        }
        return n;
    }

```

Uses `readobject()` 157b.

5.1.4 Writing a blob

```

⟨function blobify 56d⟩≡ (282b)
    void
    blobify(Dir *d, char *path, int *mode/*OUT*/, Hash *bh/*OUT*/)
    {
        char h[64]; // header
        char *buf; // blob content
        fdt f;

```

```

int nh;

⟨blobify() sanity check d is a file entry 57a⟩
*mode = d->mode;
nh = snprintf(h, sizeof(h), "%T %lld", GBlob, d->length) + 1;
f = open(path, OREAD);
⟨blobify() sanity check f 57b⟩
buf = emalloc(d->length);
if(readn(f, buf, d->length) != d->length)
    sysfatal("could not read blob %s: %r", path);
writeobj(bh, h, nh, buf, d->length);
free(buf);
close(f);
}

```

Uses `estrdup()` 257b.

```

⟨blobify() sanity check d is a file entry 57a⟩≡ (56d)
if((d->mode & DMDIR) != 0)
    sysfatal("not file: %s", path);

```

```

⟨blobify() sanity check f 57b⟩≡ (56d)
if(f == -1)
    sysfatal("could not open %s: %r", path);

```

5.1.5 Writing a tree

```

⟨function writetree 57c⟩≡ (282b)
int
writetree(Dirent *ent, int nent, Hash *h/*OUT*/)
{
    char hdr[128];
    int nhdr;
    char *t, *txt, *etxt;
    int n;
    Dirent *d, *p;

    t = emalloc((16+256+20) * nent);
    txt = t;
    etxt = t + (16+256+20) * nent;

    /* squeeze out deleted entries */
    n = 0;
    p = ent;
    for(d = ent; d != ent + nent; d++)
        if(d->name)
            p[n++] = *d;
    nent = n;

    qsort(ent, nent, sizeof(Dirent), entcmp);

    for(d = ent; d != ent + nent; d++){
        if(strlen(d->name) >= 255)
            sysfatal("overly long filename: %s", d->name);
        t = sprintf(t, etxt, "%o %s", gitmode(d), d->name) + 1;
        memcpy(t, d->h.h, sizeof(d->h.h));
        t += sizeof(d->h.h);
    }
    nhdr = snprintf(hdr, sizeof(hdr), "%T %lld", GTree, (vlong)(t - txt)) + 1;
}

```

```

    writeobj(h, hdr, nhdr, txt, t - txt);

    free(txt);
    return nent;
}

```

Uses GBlob 33a and writeobj() 56c.

```

⟨function gitmode 58a⟩≡ (282b)
int
gitmode(Dirent *e)
{
    ⟨gitmode() if symlink 236c⟩
    ⟨gitmode() if submodule 237c⟩
    else if(e->mode & DMDIR)
        return 0040000;
    else if(e->mode & 0100)
        return 0100755;
    else
        return 0100644;
}

```

5.1.6 Writing a commit

```

⟨globals parents (save.c) 58b⟩≡ (282b)
Hash    parents[Maxparents];
int nparents;

```

```

⟨const Maxparents 58c⟩≡ (282b)
Maxparents = 16,

```

```

⟨function mkcommit 58d⟩≡ (282b)
void
mkcommit(Hash *c/*OUT*/, vlong date, Hash tree)
{
    char h[64];
    int nh;
    char *s;
    int ns, i;
    Fmt f;

    fmtstrinit(&f);
    fmtprint(&f, "tree %H\n", tree);
    for(i = 0; i < nparents; i++)
        fmtprint(&f, "parent %H\n", parents[i]);
    fmtprint(&f, "author %s <%s> %lld +0000\n", authorname, authoremail, date);
    fmtprint(&f, "committer %s <%s> %lld +0000\n", committername, committeremail, date);
    fmtprint(&f, "\n");
    fmtprint(&f, "%s", commitmsg);
    s = fmtstrflush(&f);

    ns = strlen(s);
    nh = snprintf(h, sizeof(h), "%T %d", GCommit, ns) + 1;
    writeobj(c, h, nh, s, ns);
    free(s);
}

```

5.2 References

5.3 Index

Chapter 6

git/conf

6.1 Repository configuration: .git/config

6.2 User configuration: ~/lib/git/config

6.3 Usage

```
<function usage 60a>≡ (266a)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-f file] [-r] keys..\n", argv0);
    fprintf(STDERR, "\t-f:    use file 'file' (default: .git/config)\n");
    fprintf(STDERR, "\t r:    print repository root\n");
    exits("usage");
}
```

6.4 main()

```
<function main 60b>≡ (266a)
void
main(int argc, char **argv)
{
    char repo[512];
    int nrel;
    <main() (conf.c) other locals 62e>

    ARGBEGIN{
        <main() (conf.c) command line processing 62c>
        default:    usage();          break;
    }ARGEND;

    gitinit(repo, sizeof(repo), &nrel);

    <main() (conf.c) if findroot 62d>
    // else
    <main() (conf.c) set default file 61d>

    for(i = 0; i < argc; i++){
        <main() (conf.c) show config entry for argv[i] 62f>
    }
```

```

    exits(nil);
}

<signature gitinit 61a>≡ (270)
void gitinit(char*, int, int*);

<global file 61b>≡ (266a)
// array<ref_own<filename> (length=nfile)
char *file[32];

<global nfile 61c>≡ (266a)
int nfile;

<main() (conf.c) set default file 61d>≡ (60b)
if(nfile == 0){
    if((file[nfile++] = smprint("%s/.git/config", repo)) == nil)
        sysfatal("smprint: %r");
    if((p = getenv("home")) != nil)
        if((file[nfile++] = smprint("%s/lib/git/config", p)) == nil)
            sysfatal("smprint: %r");
    file[nfile++] = "/lib/git/config";
}

```

6.5 gitinit() and findrepo()

```

<function gitinit 61e>≡ (284b)
/// conf.c:main | fs.c:main | ... -> <>
void
gitinit(char *root, int nroot, int *nrel)
{
    char repo[512] = ".git";
    Dir *d;

    <gitinit() initializations 61f>
    if(root != nil){
        findrepo(root, nroot, nrel);
        snprintf(repo, sizeof(repo), "%s/.git", root);
    }
    d = dirstat(repo);
    if(d == nil)
        sysfatal("stat %s: %r", repo);
    gitdirmode = d->mode & 0777;
    free(d);
}

```

Uses findrepo() 62a and gitdirmode.

```

<gitinit() initializations 61f>≡ (61e)
<gitinit() fntinstall calls 31b>
<gitinit() libflate initializations 46a>
<gitinit() set authorpat 49a>
<gitinit() set objcache 248b>

<global gitdirmode 61g>≡ (284b)
int gitdirmode = -1;

```

```

⟨function findrepo 62a⟩≡ (284b)
/* Finds the directory containing the git repo. */
static void
findrepo(char *buf, int nbuf, int *nrel)
{
    char *p;
    char *suffix = "/.git/HEAD";

    if(getwd(buf, nbuf - strlen(suffix) - 1) == nil)
        sysfatal("getwd: %r");

    *nrel = 0;
    for(p = buf + strlen(buf); p != nil; p = strrchr(buf, '/')){
        strcpy(p, suffix);
        if(access(buf, AEXIST) == 0){
            p[p == buf] = '\0';
            return;
        }
        // else
        *nrel += 1;
        *p = '\0';
    }
    sysfatal("not a git repository");
}

```

6.6 git/conf -r, printing the project root

```

⟨global findroot 62b⟩≡ (266a)
bool findroot;

```

```

⟨main() (conf.c) command line processing 62c⟩≡ (60b) 63c▷
case 'r': findroot=true; break;

```

```

⟨main() (conf.c) if findroot 62d⟩≡ (60b)
if(findroot){
    print("%s\n", repo);
    exits(nil);
}

```

6.7 git/conf <key>, looking up a key

```

⟨main() (conf.c) other locals 62e⟩≡ (60b)
char *p, *s;
int i, j;

```

```

⟨main() (conf.c) show config entry for argv[i] 62f⟩≡ (60b)
// syntax is git/fs <sect>.<key> or just <key>
p = strchr(argv[i], '.');
if(p == nil){
    s = nil;
    p = argv[i];
}else{
    *p = '\0';
    p++;
    s = smprint("[%s]", argv[i]);
}

```

```

for(j = 0; j < nfile; j++)
    if(showconf(file[j], s, p))
        break;

```

<function showconf 63a>≡ (266a)

```

static bool
showconf(char *cfg, char *sect, char *key)
{
    Biobuf *f;
    bool found, foundsect;
    char *ln, *p;
    int nsect, nkey;

    f = Bopen(cfg, OREAD);
    if(f == nil)
        return false;

    found = false;
    nsect = sect ? strlen(sect) : 0;
    nkey = strlen(key);
    foundsect = (sect == nil);

    while((ln = Brdstr(f, '\n', 1)) != nil){
        p = strip(ln);
        if(*p == '[' && sect){
            foundsect = strncmp(sect, ln, nsect) == 0;
        }else if(foundsect && strncmp(p, key, nkey) == 0){
            p = strip(p + nkey);
            if(*p != '=')
                continue;
            p = strip(p + 1);
            print("%s\n", p);
            found = true;
            if(!showall){
                free(ln);
                goto done;
            }
        }
        free(ln);
    }
done:
    return found;
}

```

Uses showall 63b.

6.8 Advanced features

6.8.1 git/conf -a, showing all values

<global showall 63b>≡ (266a)

```

bool showall;

```

<main() (conf.c) command line processing 63c>+≡ (60b) <62c 64>

```

case 'a': showall=true; break;

```

6.8.2 git/conf -f <config>, non-standard config file path

<main() (conf.c) command line processing 64>+≡

(60b) <63c

```
case 'f':
    if(nfile == nelem(file))
        sysfatal("too many configs");
    file[nfile++] = EARGF(usage());
    break;
```

Chapter 7

git/fs

7.1 Introduction

7.2 Usage

```
<function usage (git9/fs.c) 65a>≡ (267)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-d]\n", argv0);
    fprintf(STDERR, "\t-d:    debug\n");
    exits("usage");
}
```

7.3 main()

```
<function main (git9/fs.c) 65b>≡ (267)
void
main(int argc, char **argv)
{
    char repo[512];
    int nelt;
    Dir *d;

    gitinit(repo, sizeof(repo), &nelt);
    // !!chdir!!
    if(chdir(repo) == ERROR_NEG1)
        sysfatal("chdir: %r");

    ARGBEGIN{
<main() (fs.c) command line processing 86b>
        default: usage(); break;
    }ARGEND;
    if(argc != 0)
        usage();

    // can use just relative .git path because of chdir() above
    d = dirstat(".git");
    if(d == nil)
        sysfatal("dirstat .git: %r");
    username = strdup(d->uid);
    groupname = strdup(d->gid);
}
```

```

    free(d);

    <main() (fs.c) set branches 66e>
    postmountsrv(&gitsrv, nil, mntpt, MCREATE);
    exits(nil);
}

```

```

<global username 66a>≡ (267)
char *username;

```

```

<global groupname 66b>≡ (267)
char *groupname;

```

```

<global mntpt 66c>≡ (267)
// can be changed by git/fs -m
char *mntpt = ".git/fs";

```

Uses mntpt 66c.

```

<global branches 66d>≡ (267)
// growing_array<ref_own<string>>
char **branches = nil;

```

Uses branches 66d.

```

<main() (fs.c) set branches 66e>≡ (65b)
branches = emalloc(sizeof(char*));
branches[0] = nil;

```

```

<global gitsrv 66f>≡ (267)
Srv gitsrv = {
    // will attach and set gitdir too
    .attach=gitattach,
    .walk1=gitwalk1,
    .clone=gitclone,
    .open=gitopen,
    .read=gitread,
    .stat=gitstat,
    .destroyfid=gitdestroyfid,
};

```

Uses gitattach(), gitclone(), gitdestroyfid(), gitopen(), gitread(), gitstat(), and gitwalk1().

7.4 Data structures

7.4.1 Qxxx and qroot

```

<enum Qxxx 66g>≡ (267)
enum Qxxx {
    Qroot,

    Qhead,
    Qbranch,
    Qobject,
    Qctl,

    // object/xxx/
    Qtree,
    Qcommit,
    Qmsg,
    Qparent,
};

```

```

        Qhash,
        Qauthor,
        Qcommitter,

        Qmax,
        Internal=1<<7,
};

```

```

⟨constant qroot 67a⟩≡ (267)
char *qroot[] = {
    "HEAD",
    "branch",
    "object",
    "ctl",
};

```

7.4.2 Gitaux and Crumb

```

⟨struct Gitaux 67b⟩≡ (267)
struct Gitaux {
    int ncrumb;
    // growing_array<ref_own<Crumb>> len = ncrumb
    Crumb *crumb;

    ⟨Gिताux object dir fields 67e⟩
    ⟨Gिताux other fields 79c⟩
};

```

```

⟨struct Crumb 67c⟩≡ (267)
struct Crumb {
    char *name;

    Qid qid;
    int mode;
    vlong mtime;

    // option<shared_ref<Object>>
    Object *obj;
};

```

```

⟨function crumb 67d⟩≡ (267)
static Crumb*
crumb(Gिताux *aux, int n)
{
    if(n < aux->ncrumb)
        return &aux->crumb[aux->ncrumb - n - 1];
    return nil;
}

```

7.4.3 Objlist

```

⟨Gिताux object dir fields 67e⟩≡ (67b)
/* For listing object dir */
// ref_own<Objlist> ??
Objlist *ols;
Object *olslast;

```

```

<struct Objlist 68a>≡ (270)
struct Objlist {
    int idx;

    // option<fdt> (None = -1)
    fdt fd;
    // ??
    int state;
    int stage;

    // .git/objects/
    // array<ref_own<Dir>> (len = ntop)
    Dir *top;
    int ntop;
    int topidx;

    // ??
    // array<ref_own<Dir>> (len = nloose)
    Dir *loose;
    int nloose;
    int looseidx;

    // .git/objects/pack/
    // option<array<ref_own<Dir>> (len = npack)
    Dir *pack;
    int npack;
    int packidx;

    int nent;
    int entidx;
};

```

```

<function olsfree 68b>≡ (276a)
void
olsfree(Objlist *ols)
{
    if(ols == nil)
        return;
    if(ols->fd != -1)
        close(ols->fd);
    free(ols->top);
    free(ols->loose);
    free(ols->pack);
    free(ols);
}

```

7.4.4 Uqid and Cache

```

<struct Uqid 68c>≡ (267)
struct Uqid {
    vlong    uqid;

    vlong    ppath;
    vlong    oid;

    int t;
    int idx;
};

```

```

<struct Cache 69a>≡ (267)
struct Cache {
    // growing_array<ref_own<Uqid>> (len = n)
    Uqid *cache;
    int n;
    int max;
};

```

```

<global uqidcache 69b>≡ (267)
// hash<qid, Uqid>
Cache uqidcache[512];

```

```

<function qpath 69c>≡ (267)
vlong
qpath(Crumb *p, int idx, vlong id, vlong t)
{
    int h, i;
    vlong pp;
    Cache *c;
    Uqid *u;

    pp = p ? p->qid.path : 0;
    h = (pp*333 + id*7 + t) & (nelem(uqidcache) - 1);
    c = &uqidcache[h];
    u = c->cache;
    for(i=0; i <c->n ; i++){
        if(u->ppath == pp && u->oid == id && u->t == t && u->idx == idx)
            return (u->uqid << 8) | t;
        u++;
    }
    if(c->n == c->max){
        c->max += c->max/2 + 1;
        c->cache = erealloc(c->cache, c->max*sizeof(Uqid));
    }
    nextqid++;
    c->cache[c->n] = (Uqid){nextqid, pp, id, t, idx};
    c->n++;
    return (nextqid << 8) | t;
}

```

Uses nextqid and uqidcache 69b.

```

<global nextqid 69d>≡ (267)
vlong nextqid = Qmax;

```

```

<macro QDIR 69e>≡ (270)
#define QDIR(qid) ((int)(qid)->path & (0xff))

```

7.5 gitxxx() skeletons

7.5.1 gitattach() and gitdir

```

<global gitdir 69f>≡ (267)
char gitdir[512];

```

```

⟨function gitattach 70a⟩≡ (267)
static void
gitattach(Req *r)
{
    Gitaux *aux;
    Dir *d;

    d = dirstat(".git");
    if(d == nil)
        sysfatal("git/fs: %r");
    if(getwd(gitdir, sizeof(gitdir)) == nil)
        sysfatal("getwd: %r");

    aux = emalloc(sizeof(Gitaux));
    aux->crumb = emalloc(sizeof(Crumb));
    aux->ncrumb = 1;
    ⟨gitattach() set aux->crumb 70b⟩

    r->ofcall.qid = (Qid){Qroot, 0, QTDIR};
    r->fid->qid = r->ofcall.qid;
    r->fid->aux = aux;

    respond(r, nil);
}

```

Uses Qroot-45 66g and gitdir.

```

⟨gitattach() set aux->crumb 70b⟩≡ (70a)
aux->crumb[0].qid = (Qid){Qroot, 0, QTDIR};
aux->crumb[0].obj = nil;
aux->crumb[0].mode = DMDIR | 0555;
aux->crumb[0].mtime = d->mtime;
aux->crumb[0].name = estrdup("/");

```

Uses estrdup() 257b.

7.5.2 gitclone() and gitdestroyfid()

```

⟨function gitclone 70c⟩≡ (267)
static char*
gitclone(Fid *o, Fid *n)
{
    Gitaux *aux, *oaux;
    int i;

    oaux = o->aux;
    aux = emalloc(sizeof(Gitaux));

    aux->ncrumb = oaux->ncrumb;
    aux->crumb = eamalloc(oaux->ncrumb, sizeof(Crumb));
    for(i = 0; i < aux->ncrumb; i++){
        aux->crumb[i] = oaux->crumb[i];
        aux->crumb[i].name = estrdup(oaux->crumb[i].name);
        if(oaux->crumb[i].obj)
            aux->crumb[i].obj = ref(oaux->crumb[i].obj);
    }
    if(oaux->refpath)
        aux->refpath = strdup(oaux->refpath);
    aux->qdir = oaux->qdir;
    n->aux = aux;
}

```

```

    return nil;
}

```

Uses `eamalloc()`, `estrdup()` 257b, and `ref()` 36c.

<function gitdestroyfid 71a>≡ (267)

```

static void
gitdestroyfid(Fid *f)
{
    Gitaux *aux;
    int i;

    if((aux = f->aux) == nil)
        return;
    for(i = 0; i < aux->ncrumb; i++){
        if(aux->crumb[i].obj)
            unref(aux->crumb[i].obj);
        free(aux->crumb[i].name);
    }
    olsfree(aux->ols);
    free(aux->refpath);
    free(aux->crumb);
    free(aux);
}

```

Uses `olsfree()` 68b and `unref()` 34a.

7.5.3 gitopen()

<function gitopen 71b>≡ (267)

```

static void
gitopen(Req *r)
{
    Gitaux *aux;
    Crumb *c;

    aux = r->fid->aux;
    c = crumb(aux, 0);
    switch(r->ifcall.mode&3){
    case OREAD:
    case ORDWR:
        respond(r, nil);
        break;
    case OWRITE:
        respond(r, Eperm);
        break;
    case OEXEC:
        if((c->mode & 0111) == 0)
            respond(r, Eperm);
        else
            respond(r, nil);
        break;
    default:
        respond(r, "botched mode");
        break;
    }
}

```

Uses `Eperm-59` and `crumb()` 67d.

7.5.4 gitstat()

```
<function gitstat 72a>≡ (267)
static void
gitstat(Req *r)
{
    Gitaux *aux;
    Crumb *c;

    aux = r->fid->aux;
    c = crumb(aux, 0);

    r->d.uid = estrdup9p(username);
    r->d.gid = estrdup9p(groupname);
    r->d.muid = estrdup9p(username);

    r->d.qid = r->fid->qid;
    r->d.mtime = c->mtime;
    r->d.atime = c->mtime;
    r->d.mode = c->mode;

    if(c->obj)
        obj2dir(&r->d, c, c->obj, c->name);
    else
        r->d.name = estrdup9p(c->name);
    respond(r, nil);
}
```

Uses `crumb()` 67d, `groupname`, `obj2dir()` 72b, and `username`.

```
<function obj2dir 72b>≡ (267)
static void
obj2dir(Dir *d, Crumb *c, Object *o, char *name)
{
    d->qid = c->qid;
    d->atime = c->mtime;
    d->mtime = c->mtime;
    d->mode = c->mode;

    d->name = estrdup9p(name);
    d->uid = estrdup9p(username);
    d->gid = estrdup9p(groupname);
    d->muid = estrdup9p(username);

    if(o->type == GBlob || o->type == GTag){
        d->qid.type = 0;
        d->mode &= 0777;
        d->length = o->size;
    }
}
```

Uses `GBlob` 33a, `GTag` 177b, `groupname`, and `username`.

7.5.5 gitread()

```
<function gitread 72c>≡ (267)
static void
gitread(Req *r)
{
    char buf[256], *e;
    Gitaux *aux;
```

```

Object *o;
Qid *q;

aux = r->fid->aux;
q = &r->fid->qid;
o = crumb(aux, 0)->obj;
e = nil;

switch(QDIR(q)){
<gitread() switch QDIR(q) cases 74c>
default:
    e = Egreg;
}
respond(r, e);
}

```

Uses QDIR, Qroot-45 66g, and crumb() 67d.

7.5.6 gitwalk1()

```

<function gitwalk1 73>≡ (267)
static char*
gitwalk1(Fid *fid, char *name, Qid *q)
{
    char path[128];
    Gitaux *aux;
    Crumb *c, *o;
    char *e;
    Dir *d;
    Hash h;

    e = nil;
    aux = fid->aux;

    q->vers = 0;
    <gitwalk1() if dotdot 74a>
    // else

    aux->crumb = realloc(aux->crumb, (aux->ncrumb + 1) * sizeof(Crumb));
    aux->ncrumb++;

    c = crumb(aux, 0);
    o = crumb(aux, 1);
    memset(c, 0, sizeof(Crumb));
    c->mode = o->mode;
    c->mtime = o->mtime;
    c->obj = o->obj ? ref(o->obj) : nil;

    switch(QDIR(&fid->qid)){
    <gitwalk1() switch QDIR(&fid->qid) cases 75a>
    default:
        return Egreg;
    }

    c->name = estrdup(name);
    c->qid = *q;
    fid->qid = *q;
    return e;
}

```

Uses QDIR, Qroot-45 66g, crumb() 67d, estrdup() 257b, and ref() 36c.

```

⟨gitwalk1() if dotdot 74a⟩≡ (73)
    if(strcmp(name, "..") == 0){
        popcrumb(aux);
        c = crumb(aux, 0);
        *q = c->qid;
        fid->qid = *q;
        return nil;
    }

```

Uses crumb() 67d.

```

⟨function popcrumb 74b⟩≡ (267)
    static void
    popcrumb(Gitaux *aux)
    {
        Crumb *c;

        if(aux->ncrumb > 1){
            c = crumb(aux, 0);
            free(c->name);
            unref(c->obj);
            aux->ncrumb--;
        }
    }

```

Uses crumb() 67d and unref() 34a.

7.6 .git/fs/

```

⟨gitread() switch QDIR(q) cases 74c⟩≡ (72c) 75b▷
    case Qroot:
        dirread9p(r, rootgen, aux);
        break;

```

Uses Qctl-49 66g.

```

⟨function rootgen 74d⟩≡ (267)
    static errorneg1
    rootgen(int i, Dir *d, void *p)
    {
        Crumb *c;

        c = crumb(p, 0);
        if (i >= nelem(qroot))
            return ERROR_NEG1;

        d->mode = 0555 | DMDIR;
        d->name = estrdup9p(qroot[i]);
        d->qid.vers = 0;
        d->qid.type = strcmp(qroot[i], "ctl") == 0 ? 0 : QTDIR;
        d->qid.path = qpath(nil, i, i, Qroot);
        d->uid = estrdup9p(username);
        d->gid = estrdup9p(groupname);
        d->muid = estrdup9p(username);
        d->mtime = c->mtime;
        return OK_0;
    }

```

Uses Qroot-45 66g, crumb() 67d, groupname, qpath() 69c, qroot, and username.

`<gitwalk1() switch QDIR(&fid->qid) cases 75a>≡ (73) 76a▷`

```
case Qroot:
    if(strcmp(name, "ctl") == 0){
        *q = (Qid){Qctl, 0, 0};
        c->mode = 0644;
        <gitwalk1() when Qroot case, switch on name other cases 76b>
    }else{
        e = Eexist;
    }
    break;
```

Uses Qctl-49 66g.

7.6.1 .git/fs/ctl

`<gitread() switch QDIR(q) cases 75b>+≡ (72c) <74c 76c▷`

```
case Qctl:
    e = readctl(r);
    break;
```

Uses Qobject-48 66g.

`<function readctl 75c>≡ (267)`

```
static char *
readctl(Req *r)
{
    fdt fd;
    int n;
    char data[1024], ref[512];
    char *s, *e;

    fd = open(".git/HEAD", OREAD);
    <readctl() sanity check fd 75d>
    /* empty HEAD is invalid */
    n = readn(fd, ref, sizeof(ref) - 1);
    <readctl() sanity check n 75e>
    close(fd);

    s = ref;
    ref[n] = '\0';
    if(strncmp(s, "ref:", 4) == 0)
        s += 4;
    while(*s == ' ' || *s == '\t')
        s++;
    if((e = strchr(s, '\n')) != nil)
        *e = '\0';
    if(strstr(s, "refs/") == s)
        s += strlen("refs/");

    snprintf(data, sizeof(data), "branch %s\nrepo %s\n", s, gitdir);
    readstr(r, data);
    return nil;
}
```

Uses gitdir.

`<readctl() sanity check fd 75d>≡ (75c)`

```
if(fd == ERROR_NEG1)
    return Erepo;
```

`<readctl() sanity check n 75e>≡ (75c)`

```
if(n <= 0)
    return Erepo;
```

`<gitwalk1() switch QDIR(&fid->qid) cases 76a)>+≡ (73) <75a 77b>`

```
case Qctl:
    return Enodir;
```

Uses Qobject-48 66g.

7.6.2 .git/fs/object/

`<gitwalk1() when Qroot case, switch on name other cases 76b)>≡ (75a) 83f>`

```
}else if(strcmp(name, "object") == 0){
    *q = (Qid){Qobject, 0, QTDIR};
    c->mode = DMDIR | 0555;
```

`<gitread() switch QDIR(q) cases 76c)>+≡ (72c) <75b 78e>`

```
case Qobject:
    if(o)
        objread(r, aux);
    else
        dirread9p(r, objgen, aux);
    break;
```

Uses Qtree-50 66g, objgen() 76d, and objread().

objgen()

`<function objgen 76d)>≡ (267)`

```
static int
objgen(int i, Dir *d, void *p)
{
    Gitaux *aux;
    Object *o;
    Crumb *c;
    char name[64];
    Objlist *ols;
    Hash h;

    aux = p;
    c = crumb(aux, 0);

    if(!aux->ols)
        aux->ols = mkols();
    ols = aux->ols;
    o = nil;

    /* We tried to sent it, but it didn't fit */
    if(aux->olslast && ols->idx == i + 1){
        snprintf(name, sizeof(name), "%H", aux->olslast->hash);
        obj2dir(d, c, aux->olslast, name);
        return OK_0;
    }
    // else
    while(ols->idx <= i){
        if(olsnext(ols, &h) == -1)
            return ERROR_NEG1;
        o = readobject(h);
        if(o == nil){
            fprintf(STDERR, "corrupt object %H\n", h);
            return ERROR_NEG1;
        }
    }
}
```

```

}
if(o != nil){
    snprintf(name, sizeof(name), "%H", o->hash);
    obj2dir(d, c, o, name);
    unref(aux->olslast);
    aux->olslast = ref(o);
    return OK_0;
}
// else
return ERROR_NEG1;
}

```

Uses `crumb()` 67d, `mkols()` 77a, `obj2dir()` 72b, `olsnext()` 275, `readobject()` 157b, `ref()` 36c, and `unref()` 34a.

`mkols()`

```

<function mkols 77a>≡ (276a)
Objlist*
mkols(void)
{
    Objlist *ols;

    ols = emalloc(sizeof(Objlist));
    ols->ntop = slurpdir(".git/objects", &ols->top);
    if(ols->ntop == ERROR_NEG1)
        sysfatal("read top level: %r");
    ols->npack = slurpdir(".git/objects/pack", &ols->pack);
    if(ols->npack == ERROR_NEG1)
        ols->pack = nil;
    ols->fd = -1;
    return ols;
}

```

Uses `slurpdir()` 260.

`gitwalk1()` for `Qobject`

```

<gitwalk1() switch QDIR(&fid->qid) cases 77b>+≡ (73) <76a 79d>
case Qobject:
    if(c->obj){
        e = objwalk1(q, o->obj, o, c, name, Qobject, aux);
    }else{
        if(hparse(&h, name) == ERROR_NEG1)
            return Ebadobj;
        c->obj = readobject(h);
        if(c->obj == nil)
            return Ebadobj;
        if(c->obj->type == GBlob || c->obj->type == GTag){
            c->mode = 0644;
            q->type = 0;
        }else{
            c->mode = DMDIR | 0755;
            q->type = QTDIR;
        }
        q->path = qpath(o, Qobject, c->obj->id, Qobject);
        q->vers = 0;
    }
    break;

```

Uses `Ebadobj`-67, `GBlob` 33a, `GTag` 177b, `Qobject`-48 66g, `Qtree`-50 66g, `hparse()` 31d, `objwalk1()`, `qpath()` 69c, and `readobject()` 157b.

objwalk1()

```
<function objwalk1 78a>≡ (267)
static char *
objwalk1(Qid *q, Object *o, Crumb *p, Crumb *c, char *name, vlong qdir, Gitaux *aux)
{
    Object *w, *l;
    char *e;
    int i, m;

    w = nil;
    e = nil;
    if(!o)
        return Eexist;
    switch(o->type){
<objwalk1() switch o->type cases 78d>
    }
    return e;
}
```

Uses Eexist-60 and GBlob 33a.

7.6.3 .git/fs/object/<blobid>

```
<function objread 78b>≡ (267)
static void
objread(Req *r, Gitaux *aux)
{
    Object *o;

    o = crumb(aux, 0)->obj;
    switch(o->type){
<objread() switch o->type cases 78c>
    default:
        sysfatal("invalid object type %d", o->type);
    }
}
```

Uses GBlob 33a and crumb() 67d.

```
<objread() switch o->type cases 78c>≡ (78b) 79a▷
case GBlob:
    readbuf(r, o->data, o->size);
    break;
```

Uses GTree 33a.

```
<objwalk1() switch o->type cases 78d>≡ (78a) 79e▷
case GBlob:
    e = Enotdir;
    break;
```

Uses GTree 33a.

7.6.4 .git/fs/object/<treeid>/

```
<gitread() switch QDIR(q) cases 78e>+≡ (72c) <76c 80a▷
case Qtree:
    objread(r, aux);
    break;
```

Uses Qcommit-51 66g.

`<objread() switch o->type cases 79a>+≡ (78b) <78c 80b>`

```
case GTree:
    dirread9p(r, gtreegen, aux);
    break;
```

Uses GCommit 33a.

`<function gtreegen 79b>≡ (267)`

```
static int
gtreegen(int i, Dir *d, void *p)
{
    Object *o, *l, *e;
    Gitaux *aux;
    Crumb *c;
    int m;

    aux = p;
    c = crumb(aux, 0);
    e = c->obj;
    if(i >= e->tree->nent)
        return ERROR_NEG1;
    m = e->tree->ent[i].mode;
    <gtreegen() if submodule 237d>
    else if((o = readobject(e->tree->ent[i].h)) == nil)
        sysfatal("could not read object %H: %r", e->tree->ent[i].h);

    <gtreegen() if e->tree->ent[i].islink 236d>

    d->qid.vers = 0;
    d->qid.type = o->type == GTree ? QTDIR : 0;
    d->qid.path = qpath(c, i, o->id, aux->qdir);
    d->mode = m;
    d->atime = c->mtime;
    d->mtime = c->mtime;
    d->uid = estrdup9p(username);
    d->gid = estrdup9p(groupname);
    d->muid = estrdup9p(username);
    d->name = estrdup9p(e->tree->ent[i].name);
    d->length = o->size;
    return OK_0;
}
```

Uses GTree 33a, crumb() 67d, groupname, qpath() 69c, readobject() 157b, and username.

`<Giaux other fields 79c>≡ (67b) 85a>`

```
int qdir;
```

`<gitwalk1() switch QDIR(&fid->qid) cases 79d>+≡ (73) <77b 81a>`

```
case Qtree:
    e = objwalk1(q, o->obj, o, c, name, Qtree, aux);
    break;
```

Uses Qcommit-51 66g.

`<objwalk1() switch o->type cases 79e>+≡ (78a) <78d 81b>`

```
case GTree:
    q->type = 0;
    for(i = 0; i < o->tree->nent; i++){
        if(strcmp(o->tree->ent[i].name, name) != 0)
            continue;
        m = o->tree->ent[i].mode;
        w = readobject(o->tree->ent[i].h);
```

```

⟨objwalk1() when GTree case, if o->tree->ent[i].ismod 237e⟩
⟨objwalk1() when GTree case, if o->tree->ent[i].islink 236e⟩

```

```

if(!w)
    return Ebadobj;
q->type = (w->type == GTree) ? QTDIR : 0;
q->path = qpath(p, i, w->id, qdir);
c->mode = m;
c->mode |= (w->type == GTree) ? (DMDIR|0755) : 0644;
c->obj = w;
break;
}
if(!w)
    e = Eexist;
break;

```

Uses Ebadobj-67, Eexist-60, GCommit 33a, GTree 33a, qpath() 69c, and readobject() 157b.

7.6.5 .git/fs/object/<commitid>/

```

⟨gitread() switch QDIR(q) cases 80a⟩+≡ (72c) <78e 82b>
case Qcommit:
    objread(r, aux);
    break;

```

Uses Qparent-53 66g.

```

⟨objread() switch o->type cases 80b⟩+≡ (78b) <79a 238b>
case GCommit:
    dirread9p(r, gcommitgen, aux);
    break;

```

Uses GTag 177b.

```

⟨function gcommitgen 80c⟩≡ (267)
static int
gcommitgen(int i, Dir *d, void *p)
{
    Object *o;
    Crumb *c;

    c = crumb(p, 0);
    o = c->obj;

    d->uid = estrdup9p(username);
    d->gid = estrdup9p(groupname);
    d->muid = estrdup9p(username);
    d->mode = 0444;
    d->atime = o->commit->ctime;
    d->mtime = o->commit->ctime;
    d->qid.type = 0;
    d->qid.vers = 0;

    switch(i){
    ⟨gcommitgen() switch i cases 81c⟩
    default:
        free(d->uid);
        free(d->gid);
        free(d->muid);
        return ERROR_NEG1;
    }
}

```

```

}
// else
return OK_0;
}

```

Uses `crumb()` 67d, `groupname`, and `username`.

```

<gitwalk1() switch QDIR(&fid->qid) cases 81a>+≡ (73) <79d 82f>
case Qcommit:
    e = objwalk1(q, o->obj, o, c, name, Qcommit, aux);
    break;

```

Uses `Qmsg-52` 66g.

```

<objwalk1() switch o->type cases 81b>+≡ (78a) <79e 238c>
case GCommit:
    q->type = 0;
    c->mtime = o->commit->mtime;
    c->mode = 0644;
    assert(qdir == Qcommit || qdir == Qobject || qdir == Qtree || qdir == Qhead || qdir == Qcommitter);
    if(strcmp(name, "msg") == 0)
        q->path = qpath(p, 0, o->id, Qmsg);
    else if(strcmp(name, "parent") == 0)
        q->path = qpath(p, 1, o->id, Qparent);
    else if(strcmp(name, "hash") == 0)
        q->path = qpath(p, 2, o->id, Qhash);
    else if(strcmp(name, "author") == 0)
        q->path = qpath(p, 3, o->id, Qauthor);
    else if(strcmp(name, "committer") == 0)
        q->path = qpath(p, 3, o->id, Qcommitter);
    else if(strcmp(name, "tree") == 0){
        q->type = QTDIR;
        q->path = qpath(p, 4, o->id, Qtree);
        unref(c->obj);
        c->mode = DMDIR | gitdirmode;
        c->obj = readobject(o->commit->tree);
        if(c->obj == nil)
            sysfatal("could not read object %H: %r", o->commit->tree);
    }
    else
        e = Eexist;
    break;

```

Uses `Eexist-60`, `GTag` 177b, `Qauthor-55` 66g, `Qcommit-51` 66g, `Qcommitter-56` 66g, `Qhash-54` 66g, `Qhead-46` 66g, `Qmsg-52` 66g, `Qobject-48` 66g, `Qparent-53` 66g, `Qtree-50` 66g, `gitdirmode`, `qpath()` 69c, `readobject()` 157b, and `unref()` 34a.

7.6.6 .git/fs/object/<commitid>/tree

```

<gcommitgen() switch i cases 81c>≡ (80c) 82a>
case 0:
    d->mode = DMDIR | gitdirmode;
    d->name = estrdup9p("tree");
    d->qid.type = QTDIR;
    d->qid.path = qpath(c, i, o->id, Qtree);
    break;

```

Uses `Qtree-50` 66g, `gitdirmode`, and `qpath()` 69c.

7.6.7 .git/fs/object/<commitid>/parent

```
<gcommitgen() switch i cases 82a>+≡ (80c) <81c 82d>
    case 1:
        d->name = estrdup9p("parent");
        d->qid.path = qpath(c, i, o->id, Qparent);
        break;
```

Uses Qparent-53 66g and qpath() 69c.

```
<gitread() switch QDIR(q) cases 82b>+≡ (72c) <80a 82e>
    case Qparent:
        readcommitparent(r, o);
        break;
```

Uses Qmsg-52 66g.

```
<function readcommitparent 82c>≡ (267)
    static void
    readcommitparent(Req *r, Object *o)
    {
        char *buf, *p, *e;
        int i, n;

        /* 40 bytes per hash, 1 per nl, 1 for terminator */
        n = o->commit->nparent * (40 + 1) + 1;
        buf = emalloc(n);
        p = buf;
        e = buf + n;
        for (i = 0; i < o->commit->nparent; i++)
            p = seprint(p, e, "%H\n", o->commit->parent[i]);
        readbuf(r, buf, p - buf);
        free(buf);
    }
```

7.6.8 .git/fs/object/<commitid>/msg

```
<gcommitgen() switch i cases 82d>+≡ (80c) <82a 83a>
    case 2:
        d->name = estrdup9p("msg");
        d->qid.path = qpath(c, i, o->id, Qmsg);
        break;
```

Uses Qmsg-52 66g and qpath() 69c.

```
<gitread() switch QDIR(q) cases 82e>+≡ (72c) <82b 83b>
    case Qmsg:
        readbuf(r, o->commit->msg, o->commit->nmsg);
        break;
```

Uses Qhash-54 66g.

```
<gitwalk1() switch QDIR(&fid->qid) cases 82f>+≡ (73) <81a 84c>
    case Qmsg:
    case Qparent:
    case Qhash:
    case Qauthor:
    case Qcommitter:
        return Enodir;
```

Uses Enodir-63, Qauthor-55 66g, Qcommitter-56 66g, Qhash-54 66g, and Qhead-46 66g.

7.6.9 .git/fs/object/<commitid>/hash

```
<gcommitgen() switch i cases 83a)+≡ (80c) <82d 83c>
case 3:
    d->name = estrdup9p("hash");
    d->qid.path = qpath(c, i, o->id, Qhash);
    break;
```

Uses Qhash-54 66g and qpath() 69c.

```
<gitread() switch QDIR(q) cases 83b)+≡ (72c) <82e 83d>
case Qhash:
    snprintf(buf, sizeof(buf), "%H\n", o->hash);
    readstr(r, buf);
    break;
```

Uses Qauthor-55 66g.

7.6.10 .git/fs/object/<commitid>/author

```
<gcommitgen() switch i cases 83c)+≡ (80c) <83a>
case 4:
    d->name = estrdup9p("author");
    d->qid.path = qpath(c, i, o->id, Qauthor);
    break;
```

Uses Qauthor-55 66g and qpath() 69c.

```
<gitread() switch QDIR(q) cases 83d)+≡ (72c) <83b 83e>
case Qauthor:
    snprintf(buf, sizeof(buf), "%s\n", o->commit->author);
    readstr(r, buf);
    break;
```

Uses Qcommitter-56 66g.

7.6.11 .git/fs/object/<commitid>/committer

```
<gitread() switch QDIR(q) cases 83e)+≡ (72c) <83d 84b>
case Qcommitter:
    snprintf(buf, sizeof(buf), "%s\n", o->commit->committer);
    readstr(r, buf);
    break;
```

Uses Qhead-46 66g.

7.6.12 .git/fs/HEAD/

```
<gitwalk1() when Qroot case, switch on name other cases 83f)+≡ (75a) <76b 84d>
} else if(strcmp(name, "HEAD") == 0){
    *q = (Qid){Qhead, 0, QTDIR};
    c->mode = DMDIR | 0555;
    c->obj = readref(".git/HEAD");
```

```

<function readref 84a>≡ (267)
static Object *
readref(char *pathstr)
{
    fdt f;
    char buf[128], path[128], *p, *e;
    Hash h;
    int n;

    snprintf(path, sizeof(path), "%s", pathstr);
    while(true){
        if((f = open(path, OREAD)) == ERROR_NEG1)
            return nil;
        if((n = readn(f, buf, sizeof(buf) - 1)) == ERROR_NEG1)
            return nil;
        close(f);
        buf[n] = '\0';
        if(strncmp(buf, "ref:", 4) != 0)
            break;

        // else
        p = buf + 4;
        while(isspace(*p))
            p++;
        if((e = strchr(p, '\n')) != nil)
            *e = 0;
        snprintf(path, sizeof(path), ".git/%s", p);
    }

    if(hparse(&h, buf) == ERROR_NEG1)
        return nil;

    return readobject(h);
}

```

Uses `hparse()` 31d and `readobject()` 157b.

```

<gitread() switch QDIR(q) cases 84b>+≡ (72c) <83e 85b>
case Qhead:
    /* Empty repositories have no HEAD */
    if(o == nil)
        r->ofcall.count = 0;
    else
        objread(r, aux);
    break;

```

Uses `Qbranch-47` 66g and `objread()`.

```

<gitwalk1() switch QDIR(&fid->qid) cases 84c>+≡ (73) <82f 86a>
case Qhead:
    e = objwalk1(q, o->obj, o, c, name, Qhead, aux);
    break;

```

Uses `Qbranch-47` 66g.

7.6.13 .git/fs/branch/

```

<gitwalk1() when Qroot case, switch on name other cases 84d>+≡ (75a) <83f
}else if(strncmp(name, "branch") == 0){
    *q = (Qid){Qbranch, 0, QTDIR};
    aux->refpath = estrdup(".git/refs/");

```

```
c->mode = DMDIR | 0555;
```

Uses `estrdup()` 257b.

```
<Gitaux other fields 85a>+≡ (67b) <79c
// option<ref_own<string>>, ex: ".git/refs/" for Qbranch
char *refpath;
```

```
<gitread() switch QDIR(q) cases 85b>+≡ (72c) <84b
case Qbranch:
    if(o)
        objread(r, aux);
    else
        dirread9p(r, branchgen, aux);
    break;
```

Uses `branchgen()` 85c and `objread()`.

```
<function branchgen 85c>≡ (267)
static int
branchgen(int i, Dir *d, void *)
{
    Gitaux *aux;
    Dir *refs;
    Crumb *c;
    int n;

    aux = p;
    c = crumb(aux, 0);
    refs = nil;
    d->qid.vers = 0;
    d->qid.type = QTDIR;
    d->qid.path = qpath(c, i, branchid(aux, aux->refpath), Qbranch | Internal);
    d->mode = 0555 | DMDIR;
    d->uid = estrdup9p(username);
    d->gid = estrdup9p(groupname);
    d->muid = estrdup9p(username);
    d->mtime = c->mtime;
    d->atime = c->mtime;
    if((n = slurpdir(aux->refpath, &refs)) < 0)
        return -1;
    if(i < n){
        d->name = estrdup9p(refs[i].name);
        free(refs);
        return 0;
    }else{
        free(refs);
        return -1;
    }
}
```

Uses `Internal-58` 66g, `Qbranch-47` 66g, `branchid()` 85d, `crumb()` 67d, `groupname`, `qpath()` 69c, `slurpdir()` 260, and `username`.

```
<function branchid 85d>≡ (267)
static vlong
branchid(Gitaux *aux, char *path)
{
    int i;

    for(i = 0; branches[i]; i++)
        if(strcmp(path, branches[i]) == 0)
            goto found;
```

```

branches = realloc(branches, sizeof(char *)*(i + 2));
branches[i] = estrdup(path);
branches[i + 1] = nil;

```

```

found:
    if(aux){
        if(aux->refpath)
            free(aux->refpath);
        aux->refpath = estrdup(branches[i]);
    }
    return i;
}

```

Uses branches 66d and estrdup() 257b.

```

⟨gitwalk1() switch QDIR(&fid->qid) cases 86a)⋮≡ (73) ◁84c
    case Qbranch:
        if(strcmp(aux->refpath, ".git/refs/heads") == 0 && strcmp(name, "HEAD") == 0)
            snprintf(path, sizeof(path), ".git/HEAD");
        else
            snprintf(path, sizeof(path), "%s/%s", aux->refpath, name);
        q->type = QDIR;
        d = dirstat(path);
        if(d && d->qid.type == QDIR)
            q->path = qpath(o, Qbranch, branchid(aux, path), Qbranch);
        else if(d && (c->obj = readref(path)) != nil)
            q->path = qpath(o, Qbranch, c->obj->id, Qcommit);
        else
            e = Eexist;
        if(d != nil)
            c->mode = d->mode & ~0222;
        free(d);
        break;

```

Uses Eexist-60, Qbranch-47 66g, Qcommit-51 66g, branchid() 85d, and qpath() 69c.

7.7 git/fs -m, non-standard mount point

```

⟨main() (fs.c) command line processing 86b)≡ (65b) 253b▷
    case 'm':
        mntpt = EARGF(usage());
        break;

```

Chapter 8

git/query

8.1 Usage

```
<function usage (git9/query.c) 87a>≡ (280)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-pcr] query\n", argv0);
    exits("usage");
}
```

8.2 Commit expressions

8.3 main()

```
<function main (git9/query.c) 87b>≡ (280)
void
main(int argc, char **argv)
{
    char repo[512];
    int nrel;
    // ref_own<string>, question
    char *query;
    // array<Hash>, answer
    Hash *h;
    <main() (query.c) other locals 88a>

    ARGBEGIN{
        <main() (query.c) command line processing 98c>
        default:  usage();  break;
    }ARGEND;
    if(argc == 0)
        usage();

    <main() (query.c) fmtinstall 100a>

    gitinit(repo, sizeof(repo), &nrel);
    // !! chdir !!
    if(chdir(repo) == ERROR_NEG1)
        sysfatal("chdir: %r");

    <main() (query.c) set objpfx 101d>
```

```

// query
⟨main() (query.c) set query derived from argv 88b⟩
// answer
n = resolverefs(&h, query);
free(query);
⟨main() (query.c) sanity check n result 88c⟩

⟨main() (query.c) if changes 98d⟩
else{
    p = "";
    ⟨main() (query.c) adjust p if fullpath 101e⟩
    for(j = 0; j < n; j++) {
        i = j;
        ⟨main() (query.c) adjust i if reverse 101h⟩
        print("%s%H\n", p, h[i]);
    }
}
exits(nil);
}

```

Uses Pfmt() 100b, changes 98b, and fullpath 101a.

```

⟨main() (query.c) other locals 88a⟩≡ (87b) 101c▷
char *p, *e;
int i, j, n;

```

```

⟨main() (query.c) set query derived from argv 88b⟩≡ (87b)
// p = concat(argv, " ")
for(i = 0, n = 0; i < argc; i++)
    n += strlen(argv[i]) + 1;
query = emalloc(n+1);
p = query;
e = query + n;
for(i = 0; i < argc; i++)
    p = sprintf(p, e, "%s ", argv[i]);

```

Uses resolverefs().

```

⟨main() (query.c) sanity check n result 88c⟩≡ (87b)
if(n == ERROR_NEG1)
    sysfatal("resolve: %r");

```

8.4 Data structures

8.4.1 Eval

```

⟨struct Eval 88d⟩≡ (281)
struct Eval {
    // ref<string> (owner = main() query)
    char *str;
    // pointer in str as we parse it
    char *p;

    // stack<ref_own<Object>>, len = nstk
    Object **stk;
    int nstk;
    int stksz;
};

```

8.5 resolverefs()

```
<function resolverefs 89a>≡ (281)
// main(get.c) -> <>
errorneg1
resolverefs(Hash **r/*OUT*/, char *ref)
{
    Eval ev;
    Hash *h;
    int i;
    errorneg1 res;

    res = evalexpr(&ev, ref);

    if(res == ERROR_NEG1){
        free(ev.stk);
        return ERROR_NEG1;
    }
    h = eamalloc(ev.nstk, sizeof(Hash));
    for(i = 0; i < ev.nstk; i++)
        h[i] = ev.stk[i]->hash;
    *r = h;
    free(ev.stk);
    return ev.nstk;
}
```

Uses eamalloc() and evalexpr().

8.6 resolveref()

```
<function resolveref 89b>≡ (281)
/// main(walk.c -b) | main(save.c -p) | (main(log.c) -> showcommits) -> <>
errorneg1
resolveref(Hash *r/*OUT*/, char *ref)
{
    Eval ev;

    if(evalexpr(&ev, ref) == ERROR_NEG1){
        free(ev.stk);
        return ERROR_NEG1;
    }
    if(ev.nstk != 1){
        werrstr("ambiguous ref expr");
        free(ev.stk);
        return ERROR_NEG1;
    }
    *r = ev.stk[0]->hash;
    free(ev.stk);
    return OK_0;
}
```

Uses evalexpr().

8.7 Commit expression evaluator

```
<function evalexpr 89c>≡ (281)
errorneg1
evalexpr(Eval *ev, char *ref)
```

```

{
  memset(ev, 0, sizeof(*ev));
  ev->str = ref;
  ev->p = ref;

  while(true){
    if(evalpostfix(ev) == ERROR_NEG1)
      return ERROR_NEG1;
    if(ev->p[0] == '\0')
      return OK_0;
    else if(take(ev, ":") || take(ev, "..")){
      if(evalpostfix(ev) == ERROR_NEG1)
        return ERROR_NEG1;
      if(ev->p[0] != '\0'){
        werrstr("junk at end of expression");
        return ERROR_NEG1;
      }
      return range(ev);
    }
  }
}

```

Uses evalpostfix() and take() 90a.

```

⟨function take 90a⟩≡ (281)
bool
take(Eval *ev, char *m)
{
  int l;

  l = strlen(m);
  if(strncmp(ev->p, m, l) != 0)
    return false;
  ev->p += l;
  return true;
}

```

8.7.1 evalpostfix()

```

⟨function evalpostfix 90b⟩≡ (281)
errorneg1
evalpostfix(Eval *ev)
{
  char name[256];
  Hash h;
  Object *o;
  errorneg1 ret;

  eatspace(ev);
  if(!word(ev, name, sizeof(name))){
    werrstr("expected name in expression");
    return ERROR_NEG1;
  }
  ret = readref(&h, name);
  ⟨evalpostfix() sanity check ret invalid ref 91b⟩
  ⟨evalpostfix() if Zhash 91e⟩
  else {
    o = readobject(h);
    ⟨evalpostfix() sanity check o invalid ref 91c⟩
  }
}

```

```

push(ev, o);

while(true){
    eatspace(ev);
    switch(ev->p[0]){
        case '^':
        case '~':
            ev->p++;
            ret = parent(ev);
            <evalpostfix() sanity check ret 91d>
            break;
        case '@':
            ev->p++;
            ret = lca(ev);
            <evalpostfix() sanity check ret 91d>
            break;
        default:
            goto done;
            break;
    }
}
done:
return OK_0;
}

```

Uses `eatspace()`, `parent()` 93a, and `word()` 92a.

```

<function eatspace 91a>≡ (281)
void
eatspace(Eval *ev)
{
    while(isspace(ev->p[0]))
        ev->p++;
}

```

```

<evalpostfix() sanity check ret invalid ref 91b>≡ (90b)
if(ret == ERROR_NEG1){
    werrstr("invalid ref %s", name);
    return ret;
}

```

Uses `Zhash` 30b, `hasheq()`, and `zcommit-25` 281.

```

<evalpostfix() sanity check o invalid ref 91c>≡ (90b)
if(o == nil){
    werrstr("invalid ref %s (hash %H)", name, h);
    return ERROR_NEG1;
}

```

Uses `push()`.

```

<evalpostfix() sanity check ret 91d>≡ (90)
if(ret == ERROR_NEG1)
    return ret;

```

```

<evalpostfix() if Zhash 91e>≡ (90b)
if(hasheq(&h, &Zhash))
    o = &zcommit;

```

Uses `readobject()` 157b.

8.7.2 word()

```
<function word 92a>≡ (281)
bool
word(Eval *ev, char *b, int nb)
{
    char *p, *e;
    int n;

    p = ev->p;
    for(e = p; isword(*e) && strcmp(e, "..", 2) != 0; e++)
        /* nothing */;
    /* 1 for nul terminator */
    n = e - p + 1;
    if(n >= nb)
        n = nb;
    snprintf(b, n, "%s", p);
    ev->p = e;
    return n > 0;
}
```

Uses isword() 92b.

```
<function isword 92b>≡ (281)
bool
isword(char e)
{
    return isalnum(e) || e == '/' || e == '-' || e == '_' || e == '.';
}
```

8.7.3 push() and pop()

```
<function push 92c>≡ (281)
void
push(Eval *ev, Object *o)
{
    <push() realloc if needed 92d>
    ev->stk[ev->nstk++] = o;
}
```

```
<push() realloc if needed 92d>≡ (92c)
if(ev->nstk == ev->stksz){
    ev->stksz = 2*ev->stksz + 1;
    ev->stk = erealloc(ev->stk, ev->stksz*sizeof(Object*));
}
```

```
<function pop 92e>≡ (281)
Object*
pop(Eval *ev)
{
    <pop() sanity check empty stack 92f>
    return ev->stk[--ev->nstk];
}
```

```
<pop() sanity check empty stack 92f>≡ (92e)
if(ev->nstk == 0)
    sysfatal("stack underflow");
```

8.7.4 parent() and ^

```
<function parent 93a>≡ (281)
static errorneg1
parent(Eval *ev)
{
    Object *o, *p;

    o = pop(ev);
    <parent() sanity check o is a commit 93b>
    /* Special case: first commit has no parent. */
    if(o->commit->nparent == 0)
        p = emptydir();
    else {
        p = readobject(o->commit->parent[0]);
        <parent() sanity check p 93c>
    }
    push(ev, p);
    return OK_0;
}
```

Uses emptydir(), pop() 92e, push(), and readobject() 157b.

```
<parent() sanity check o is a commit 93b>≡ (93a)
if(o->type != GCommit){
    werrstr("not a commit: %H", o->hash);
    return ERROR_NEG1;
}
```

Uses GCommit 33a.

```
<parent() sanity check p 93c>≡ (93a)
if (p == nil){
    werrstr("no parent for %H", o->hash);
    return ERROR_NEG1;
}
```

8.7.5 lca() and @

```
<function lca 93d>≡ (281)
errorneg1
lca(Eval *ev)
{
    Object *a, *b;
    // array<ref<Object>>
    Object **o;
    int n;

    if(ev->nstk < 2){
        werrstr("ancestor needs 2 objects");
        return ERROR_NEG1;
    }
    n = 0;
    b = pop(ev);
    a = pop(ev);
    paint(&a->hash, 1, &b->hash, 1, &o, &n, Lca);
    if(n == 0)
        return ERROR_NEG1;
    push(ev, *o);
    free(o);
    return OK_0;
}
```

```
}
```

Uses Lca-21 95d, pop() 92e, and push().

```
<function ancestor 94a>≡ (281)
  /// sendpack -> <>
  Object*
  ancestor(Object *a, Object *b)
  {
    // array<ref<Object>>
    Object **o;
    Object *r;
    int n;

    if(paint(&a->hash, 1, &b->hash, 1, &o, &n, Lca) == ERROR_NEG1 || n == 0)
      return nil;
    r = ref(o[0]);
    free(o);
    return r;
  }
```

Uses Lca-21 95d and ref() 36c.

```
<paint() switch mode cases 94b>≡ (96) 95c>
  case Lca:
    dprint(1, "found ancestor\n");
    o = nil;
    for(i = 0; i < keep.sz; i++){
      o = keep.obj[i];
      if(o != nil && oshas(&drop, o->hash) && !oshas(&skip, o->hash))
        break;
    }
    if(i == keep.sz){
      *nres = 0;
      *res = nil;
    }else{
      *nres = 1;
      *res = eamalloc(1, sizeof(Object*));
      (*res)[0] = o;
    }
    break;
```

8.7.6 range() and '...'

```
<function range 94c>≡ (281)
  static errorneg1
  range(Eval *ev)
  {
    Object *a, *b;
    // array<ref<Object>>
    Object **o;
    int i, n;

    b = pop(ev);
    a = pop(ev);
    <range() if b has Zhash 95a>
    <range() if a has Zhash 95b>
    if(a->type != GCommit || b->type != GCommit){
      werrstr("non-commit object in range");
      return ERROR_NEG1;
    }
```

```

}

if (paint(&b->hash, 1, &a->hash, 1, &o, &n, Range) == ERROR_NEG1)
    return ERROR_NEG1;
for (i = 0; i < n; i++)
    push(ev, o[i]);
free(o);
return OK_0;
}

```

Uses GCommit 33a, Range-22 95d, pop() 92e, and push().

```

⟨range() if b has Zhash 95a⟩≡ (94c)
    if (hasheq(&b->hash, &Zhash))
        b = &zcommit;

```

Uses Zhash 30b, hasheq(), and zcommit-25 281.

```

⟨range() if a has Zhash 95b⟩≡ (94c)
    if (hasheq(&a->hash, &Zhash))
        a = &zcommit;

```

Uses Zhash 30b, hasheq(), and zcommit-25 281.

```

⟨paint() switch mode cases 95c⟩+≡ (96) <94b 164c>
    case Range:
        dprint(1, "found range\n");
        *res = eamalloc(nrange, sizeof(Object*));
        *nres = 0;
        for (i = nrange - 1; i >= 0; i--){
            o = range[i];
            if (!oshas(&drop, o->hash) && !oshas(&skip, o->hash)){
                (*res)[*nres] = o;
                (*nres)++;
            }
        }
        free(range);
        break;

```

8.8 paint()

```

⟨enum PaintMode 95d⟩≡ (281)
    enum PaintMode {
        Lca,
        Range,
        ⟨PaintMode other cases 164b⟩
    };

```

```

⟨enum PaintQcolor 95e⟩≡ (281)
    enum PaintQcolor {
        Blank,

        Keep,
        Drop,
        Skip,
    };

```

<function paint 96>≡

(281)

```
/// lca | ancestor | range -> <>
static errorneg1
paint(Hash *head, int nhead, Hash *tail, int ntail, Object ***res, int *nres, int mode)
{
    int i;
    Objset keep, drop, skip;
    Objq objq;
    Qelt e;
    Object *o, *c;
    Object **range;
    int nrange;
    int nskip;

    osinit(&keep);
    osinit(&drop);
    osinit(&skip);
    qinit(&objq);
    range = nil;
    nrange = 0;
    nskip = 0;

    for(i = 0; i < nhead; i++){
        if(hasheq(&head[i], &Zhash))
            continue;
        o = readobject(head[i]);
        if(o == nil){
            fprintf(STDERR, "warning: %H does not point at commit\n", head[i]);
            werrstr("read head %H: %r", head[i]);
            return ERROR_NEG1;
        }
        if(o->type != GCommit){
            fprintf(STDERR, "warning: %H does not point at commit\n", o->hash);
            unref(o);
            continue;
        }
        dprint(1, "init: keep %H\n", o->hash);
        qput(&objq, o, Keep);
        unref(o);
    }
    for(i = 0; i < ntail; i++){
        if(hasheq(&tail[i], &Zhash))
            continue;
        o = readobject(tail[i]);
        if(o == nil){
            werrstr("read tail %H: %r", tail[i]);
            return ERROR_NEG1;
        }
        if(o->type != GCommit){
            fprintf(STDERR, "warning: %H does not point at commit\n", o->hash);
            unref(o);
            continue;
        }
        dprint(1, "init: drop %H\n", o->hash);
        qput(&objq, o, Drop);
        unref(o);
    }

    dprint(1, "finding twixt commits\n");
    while(objq.nheap != 0 && qpop(&objq, &e)){
```

```

if(e.color == Skip)
    nskip--;
if(oshas(&skip, e.o->hash))
    continue;
// else

switch(e.color){
case Keep:
    if(oshas(&keep, e.o->hash))
        continue;
    if(oshas(&drop, e.o->hash))
        e.color = Skip;
    else if(mode == Range){
        range = earealloc(range, nrange+1, sizeof(Object*));
        range[nrange++] = e.o;
    }
    osadd(&keep, e.o);
    break;
case Drop:
    if(oshas(&drop, e.o->hash))
        continue;
    if(oshas(&keep, e.o->hash))
        e.color = Skip;
    osadd(&drop, e.o);
    break;
case Skip:
    osadd(&skip, e.o);
    break;
}
o = readobject(e.o->hash);
if(o->type != GCommit){
    werrstr("not a commit: %H", o->hash);
    goto error;
}

for(i = 0; i < o->commit->nparent; i++){
    c = readobject(e.o->commit->parent[i]);
    if(c == nil)
        goto error;
    if(c->type != GCommit){
        fprintf(STDERR, "warning: %H does not point at commit\n", c->hash);
        unref(c);
        continue;
    }
    dprint(2, "\tenqueue: %s %H\n", colors[e.color], c->hash);
    qput(&objq, c, e.color);
    unref(c);
    if(e.color == Skip)
        nskip++;
}
unref(o);
}

switch(mode){
<paint() switch mode cases 94b>
}
osclear(&keep);
osclear(&drop);
osclear(&skip);
return OK_0;

```

```

error:
    dprint(1, "paint error: %r\n");
    free(objq.heap);
    free(range);
    return ERROR_NEG1;
}

```

<constant colors 98a>≡ (281)

```

static char *colors[] = {
    [Keep] "keep",
    [Drop] "drop",
    [Blank] "blank",
    [Skip] "skip",
};

```

8.9 Advanced features

8.9.1 git/query -c, list changes between commits

<global changes(query.c) 98b>≡ (280)

```

bool changes;

```

<main() (query.c) command line processing 98c>≡ (87b) 101b▷

```

case 'c': changes=true; break;

```

<main() (query.c) if changes 98d>≡ (87b)

```

if(changes){
    for(i = 1; i < n; i++)
        diffcommits(h[0], h[i]);
}

```

<function diffcommits 98e>≡ (280)

```

void
diffcommits(Hash ah, Hash bh)
{
    Object *a, *b, *at, *bt;

    at = nil;
    bt = nil;
    if(!hasheq(&ah, &Zhash) && (a = readobject(ah)) != nil){
        if(a->type != GCommit)
            sysfatal("not commit: %H", ah);
        at = readobject(a->commit->tree);
        if(at == nil)
            sysfatal("bad hash %H", a->commit->tree);
        unref(a);
    }
    if(!hasheq(&bh, &Zhash) && (b = readobject(bh)) != nil){
        if(b->type != GCommit)
            sysfatal("not commit: %H", ah);
        bt = readobject(b->commit->tree);
        if(bt == nil)
            sysfatal("bad hash %H", b->commit->tree);
        unref(b);
    }
    difftrees(at, bt);
    unref(at);
    unref(bt);
}

```

Uses GCommit 33a, Zhash 30b, difftrees() 99, hasheq(), readobject() 157b, and unref() 34a.

<function difftrees 99)≡

(280)

```
void
difftrees(Object *a, Object *b)
{
    Dirent *ap, *bp, *ae, *be;
    int c;

    ap = ae = nil;
    bp = be = nil;
    if(a != nil){
        if(a->type != GTree)
            return;
        ap = a->tree->ent;
        ae = ap + a->tree->nent;
    }
    if(b != nil){
        if(b->type != GTree)
            return;
        bp = b->tree->ent;
        be = bp + b->tree->nent;
    }
    while(ap != ae && bp != be){
        c = entcmp(ap, bp);
        if(c == 0){
            if(ap->mode == bp->mode && hasheq(&ap->h, &bp->h))
                goto next;
            if(ap->mode != bp->mode)
                print("! %P%s\n", ap->name);
            else if(!(ap->mode & DMDIR) || !(bp->mode & DMDIR))
                print("@ %P%s\n", ap->name);
            if((ap->mode & DMDIR) && (bp->mode & DMDIR)){
                if(npath >= nelem(path))
                    sysfatal("path too deep");
                path[npath++] = ap->name;
                a = readobject(ap->h);
                if(a == nil)
                    sysfatal("bad hash %H", ap->h);
                b = readobject(bp->h);
                if(b == nil)
                    sysfatal("bad hash %H", bp->h);
                difftrees(a, b);
                unref(a);
                unref(b);
                npath--;
            }
        }
        next:
            ap++;
            bp++;
        }else if(c < 0) {
            show(ap, '-');
            ap++;
        }else if(c > 0){
            show(bp, '+');
            bp++;
        }
    }
    for(; ap != ae; ap++)
        show(ap, '-');
    for(; bp != be; bp++)
        show(bp, '+');
```

```
}
```

Uses `GTree` 33a, `difftrees()` 99, `entcmp()` 35c, `hasheq()`, `npath` 100d, `path` 100c, `readobject()` 157b, and `unref()` 34a.

```
<main() (query.c) fmtinstall 100a>≡ (87b)
    fmtinstall('P', Pfmt);
```

Uses `gitinit()` 61e and `nrel`.

```
<function Pfmt 100b>≡ (280)
    int
    Pfmt(Fmt *f)
    {
        int i, n;

        n = 0;
        for(i = 0; i < npath; i++)
            n += fmtprint(f, "%s/", path[i]);
        return n;
    }
```

Uses `npath` 100d and `path` 100c.

```
<global path(query.c) 100c>≡ (280)
    char *path[128];
```

```
<global npath(query.c) 100d>≡ (280)
    int npath;
```

```
<function show (git9/query.c) 100e>≡ (280)
    void
    show(Dirent *e, char m)
    {
        if(e->mode & DMDIR)
            showdir(e->h, e->name, m);
        else
            print("%c %P%s\n", m, e->name);
    }
```

Uses `showdir()` 100f.

```
<function showdir 100f>≡ (280)
    /// show -> <>
    void
    showdir(Hash dh, char *dname, char m)
    {
        Dirent *p, *e;
        Object *d;

        path[npath++] = dname;
        if((d = readobject(dh)) == nil)
            sysfatal("bad hash %H", dh);
        assert(d->type == GTree);
        p = d->tree->ent;
        e = p + d->tree->nent;
        for(; p != e; p++){
            if(p->ismod)
                continue;
            if(p->mode & DMDIR)
                showdir(p->h, p->name, m);
            else
                print("%c %P%s\n", m, p->name);
        }
    }
```

```

    print("%c %P\n", m);
    unref(d);
    npath--;
}

```

Uses GTree 33a, npath 100d, path 100c, readobject() 157b, showdir() 100f, and unref() 34a.

8.9.2 git/query -p, full path listing

```

⟨global fullpath(query.c) 101a⟩≡ (280)
    bool fullpath;

```

```

⟨main() (query.c) command line processing 101b⟩+≡ (87b) <98c 101g>
    case 'p':    fullpath=true; break;

```

```

⟨main() (query.c) other locals 101c⟩+≡ (87b) <88a
    char *objpfx;

```

```

⟨main() (query.c) set objpfx 101d⟩≡ (87b)
    if((objpfx = smprint("%s/.git/fs/object/", repo)) == nil)
        sysfatal("smprint: %r");

```

```

⟨main() (query.c) adjust p if fullpath 101e⟩≡ (87b)
    if (fullpath)
        p = objpfx;

```

8.9.3 git/query -r, reversed listing

```

⟨global reverse(query.c) 101f⟩≡ (280)
    bool reverse;

```

```

⟨main() (query.c) command line processing 101g⟩+≡ (87b) <101b 254a>
    case 'r':    reverse ^= 1;    break;

```

```

⟨main() (query.c) adjust i if reverse 101h⟩≡ (87b)
    if (reverse)
        i = n - 1 - j;

```

Chapter 9

git/walk

9.1 Introduction

9.2 Usage

```
<function usage (git9/walk.c) 102a>≡ (285)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-qbcI] [-f filt] [-b base] [paths...]\n", argv0);
    exits("usage");
}
```

9.3 main()

```
<global repopath(walk.c) 102b>≡ (285)
char    repopath[1024];
```

```
<global wdirpath(walk.c) 102c>≡ (285)
char    wdirpath[1024];
```

```
<global isindexed(walk.c) 102d>≡ (285)
bool    isindexed = true;
```

```
<global staleidx(walk.c) 102e>≡ (285)
bool    staleidx = false;
```

```
<global nrel(walk.c) 102f>≡ (285)
int     nrel;
```

```
<function main (git9/walk.c) 102g>≡ (285)
void
main(int argc, char **argv)
{
    Biobuf *f; // .git/INDEX9
    Biobuf *o; // STDOUT
    <main()(walk.c) other locals 52b>

    gitinit(repopath, sizeof(repopath), &nrel);
    if(getwd(wdirpath, sizeof(wdirpath)) == nil)
        sysfatal("getwd: %r");
    // !! chdir !!
    if(chdir(repopath) == ERROR_NEG1)
```

```

    sysfatal("chdir: %r");
if(access(".git/fs/ctl", AEXIST) != 0)
    sysfatal("no running git/fs");

ARGBEGIN{
⟨main() (walk.c) command line processing 103c⟩
default: usage(); break;
}ARGEND;

if(printflg == 0)
    printflg = Tflg | Aflg | Mflg | Rflg;

⟨main() (walk.c) initializations part 1 105c⟩

if(isindexed && !staleidx){
    f = Bopen(".git/INDEX9", OREAD);
    if(f == nil){
        staleidx = true;
        goto Stale;
    }
    ⟨main() (walk.c) initializations part 2, when has .git/INDEX9 52c⟩
}
else {
Stale:
    ⟨main() (walk.c) initializations part 2, when stale 113a⟩
}
⟨main() (walk.c) initializations part 3 104c⟩
o = Bfdopen(STDOUT, OWRITE);
⟨main() (walk.c) sanity check o 103a⟩

⟨main() (walk.c) walking idx and wdir 107b⟩
Bterm(o);

if(isindexed && staleidx)
    ⟨main() (walk.c) finalizations, when isindexed && staleidx 111b⟩

Nope:
    ⟨main() (walk.c) after Nope label 112b⟩
}

⟨main() (walk.c) sanity check o 103a⟩≡ (102g)
    if(o == nil)
        sysfatal("open out: %r");

⟨global quiet (walk.c) 103b⟩≡ (285)
    bool quiet;

⟨main() (walk.c) command line processing 103c⟩≡ (102g) 103d▷
    case 'q':
        quiet=true;
        break;

⟨main() (walk.c) command line processing 103d⟩+≡ (102g) <103c 112e▷
    case 'f':
        for(p = EARGF(usage()); *p; p++){
            switch(*p){
                case 'T': printflg |= Tflg; break;
                case 'A': printflg |= Aflg; break;
                case 'M': printflg |= Mflg; break;
                case 'R': printflg |= Rflg; break;
            }
        }
    }

```

```

        case 'U':  printf("U:  printflg != Uflg;  break;");
        default:  usage();          break;
    }
    break;

```

9.4 Initializations

9.4.1 The index state: idx

9.4.2 The working directory state: wdir

<globals wdir(walk.c) 104a>≡ (285)

```

// growing_array<Idxent> (used = nwdir, allocated = wdirs)
Idxent *wdir;
int wdirs;
int nwdir;

```

<global intree(walk.c) 104b>≡ (285)

```

bool intree;

```

<main() (walk.c) initializations part 3 104c>≡ (102g)

```

qsort(idx, nidx, sizeof(Idxent), idxcmp);

nwdir = 0;
wdirs = 32;
wdir = emalloc(wdirs*sizeof(Idxent));

intree = false;
if(argc == 0)
    loadwdir(".");
else for(i = 0; i < argc; i++)
    loadwdir(argrel[i]);
qsort(wdir, nwdir, sizeof(Idxent), idxcmp);

```

<function loadwdir 104d>≡ (285)

```

void
loadwdir(char *path)
{
    fdt fd;
    int i, n;
    Dir *d, *e;

    d = nil;
    e = nil;
    cleannname(path);
    if(!intree
    && strncmp(path, ".git", 4) == 0
    && (path[4] == '/' || path[4] == 0))
        return;
    // else

    fd = open(path, OREAD);
    if(fd < 0)
        goto error;
    e = dirfstat(fd);
    if(e == nil)
        fprintf(STDERR, "fstat: %r");
    if(e->qid.type & QTDIR)

```

```

        while((n = dirread(fd, &d)) > 0){
            for(i = 0; i < n; i++){
                loadent(path, &d[i], false);
                free(d);
            }
        }
    else
        loadent(path, e, true);
error:
    free(e);
    if(fd != -1)
        close(fd);
}

```

Uses `intree` and `loadent()` 105a.

```

<function loadent 105a>≡ (285)
void
loadent(char *dir, Dir *d, bool fullpath)
{
    char *path;
    Idxent *e;

    if(fullpath)
        path = estrdup(dir);
    else if((path = smprint("%s/%s", dir, d->name)) == nil)
        sysfatal("smprint: %r");

    cleannname(path);
    if(!intree && (printfg & Uflg) == 0 && !indexed(path, d->qid.type & QTDIR)){
        free(path);
        return;
    }
    if(d->qid.type & QTDIR){
        loadwdir(path);
        free(path);
    }else{
        if(nwdir == wdirsz){
            wdirsz += wdirsz/2;
            wdir = erealloc(wdir, wdirsz*sizeof(Idxent));
        }
        e = wdir + nwdir;
        e->path = path;
        e->qid = !intree? d->qid : (Qid){-1,-1,-1};
        e->mode = d->mode;
        e->order = nwdir;
        e->state = 'T';
        nwdir++;
    }
}

```

Uses `Uflg-15`, `estrdup()` 257b, `indexed()` 106b, `intree`, `loadwdir()` 104d, `nwdir` 285, `printfg`, `wdir`, and `wdirsz` 104a.

```

<main() (walk.c) other locals 105b>+≡ (102g) <52b 107a>
char **argrel;
int *argn;

```

```

<main() (walk.c) initializations part 1 105c>≡ (102g)
argrel = emalloc(argc*sizeof(char*));
argn = emalloc(argc*sizeof(int));
for(i = 0; i < argc; i++){
    argrel[i] = reporel(argv[i]);
    argn[i] = strlen(argrel[i]);
}

```

<function reporel 106a>≡ (285)

```
char*
reporel(char *s)
{
    char *p;
    int n;

    if(*s == '/')
        s = estrdup(s);
    else if((s = smprint("%s/%s", wdirpath, s)) == nil)
        sysfatal("smprint: %r");

    p = cleannname(s);
    n = strlen(repopath);
    if(strncmp(s, repopath, n) != 0)
        sysfatal("path outside repo: %s", s);
    p += n;
    if(*p == '/')
        p++;
    memmove(s, p, strlen(p)+1);
    return s;
}
```

Uses `estrdup()` 257b, `repopath`, and `wdirpath`.

<function indexed 106b>≡ (285)

```
int
indexed(char *path, int dir)
{
    int lo, hi, mid, r;

    r = -1;
    lo = 0;
    hi = nidx-1;
    while(lo <= hi){
        mid = (hi + lo) / 2;
        r = pathcmp(path, idx[mid].path, dir);
        if(r < 0)
            hi = mid-1;
        else if(r > 0)
            lo = mid+1;
        else
            break;
    }
    return r == 0;
}
```

Uses `pathcmp()` 106c.

<function pathcmp 106c>≡ (285)

```
/*
 * Tricky; we want to know if a file or dir is indexed,
 * but a dir is only indexed if we have a file with dir/
 * listed in the index.
 *
 * as a result, we need to add a virtual '/' at the end
 * of the path if we're doing it, so if we have:
 * foo.bar/x
 * foo/y
 * and we want to find out if foo is a directory we should
 * descend into, we need to compare as though foo/ ended
 * with a '/', or we'll bsearch down do foo.bar, not foo.
```

```

*
* this code resembles entcmp() in util.c, but differs
* because we're comparing whole paths.
*/
int
pathcmp(char *sa, char *sb, int sadir)
{
    unsigned a, b;

    while(true){
        a = *sa++;
        b = *sb++;
        if(a != b){
            if(a == 0 && sadir)
                a = '/';
            if(a == '/' && b == '/')
                return 0;
            return (a > b) ? 1 : -1;
        }
        if(a == 0)
            return 0;
    }
}

```

9.5 Walking

```

⟨main() (walk.c) other locals 107a⟩+≡ (102g) <105b 111a⟩
int i, j;
int c;

```

```

⟨main() (walk.c) walking idx and wdir 107b⟩≡ (102g)
i = 0;
j = 0;
while(i < nidx || j < nwdir){
    /* find the last entry we tracked for a path */
    while(i+1 < nidx && strcmp(idx[i].path, idx[i+1].path) == 0){
        staleidx = true;
        i++;
    }
    while(j+1 < nwdir && strcmp(wdir[j].path, wdir[j+1].path) == 0)
        j++;

    if(i < nidx && !pfxmatch(idx[i].path, argrel, argn, argc)){
        i++;
        continue;
    }
    if(i >= nidx)
        c = 1;
    else if(j >= nwdir)
        c = -1;
    else
        c = strcmp(idx[i].path, wdir[j].path);

    /* exists in both index and on disk */
    if(c == 0){
        ⟨main() (walk.c) when walking and exists in both index and disk 108c⟩
        i++;
        j++;
    }
}

```

```

/* only exists in index */
}else if(c < 0){
    <main() (walk.c) when walking and exists only in index 109a>
    i++;
/* only exists on disk */
}else{
    <main() (walk.c) when walking and exists only on disk 109b>
    j++;
}
}

<function pfxmatch 108a>≡ (285)
int
pfxmatch(char *p, char **pfx, int *pfxlen, int npfx)
{
    int i;

    if(p == nil)
        return 0;
    if(npfx == 0)
        return 1;
    for(i = 0; i < npfx; i++){
        if(strncmp(p, pfx[i], pfxlen[i]) != 0)
            continue;
        if(p[pfxlen[i]] == '/' || p[pfxlen[i]] == 0)
            return 1;
        if(strcmp(pfx[i], ".") == 0 || *pfx[i] == 0)
            return 1;
    }
    return 0;
}

```

9.6 Showing

```

<globals xxxstr 108b>≡ (285)
char    *rstr   = "R ";
char    *mstr   = "M ";
char    *astr   = "A ";
char    *ustr   = "U ";
char    *tstr   = "T ";

```

Uses mstr 108b and rstr 108b.

```

<main() (walk.c) when walking and exists in both index and disk 108c>≡ (107b)
    if(idx[i].state == 'R'){
        if(checkedin(&idx[i], false))
            show(o, Rflg, rstr, idx[i].path);
        else{
            idx[i].state = 'U';
            staleidx = true;
        }
    }
}
else if(idx[i].state == 'A' && !checkedin(&idx[i], true))
    show(o, Aflg, astr, idx[i].path);
else if(!samedata(&idx[i], &wdir[j]))
    show(o, Mflg, mstr, idx[i].path);
else
    show(o, Tflg, tstr, idx[i].path);

```

```

⟨main() (walk.c) when walking and exists only in index 109a⟩≡ (107b)
    if(checkedin(&idx[i], false))
        show(o, Rflg, rstr, idx[i].path);
    else{
        idx[i].state = 'U';
        staleidx = true;
    }

```

```

⟨main() (walk.c) when walking and exists only on disk 109b⟩≡ (107b)
    if(checkedin(&wdir[j], false)){
        if(samedata(nil, &wdir[j]))
            show(o, Tflg, tstr, wdir[j].path);
        else
            show(o, Mflg, mstr, wdir[j].path);
    }else if(printflg & Uflg && pfxmatch(wdir[j].path, argrel, argn, argc))
        show(o, Uflg, ustr, wdir[j].path);

```

```

⟨function show (git9/walk.c) 109c⟩≡ (285)
void
show(Biobuf *o, int flg, char *str, char *path)
{
    char *pa, *pb;
    int n;

    dirty |= flg;
    if(!quiet && (printflg & flg)){
        Bprint(o, str);
        n = nslash;
        if(n){
            for(pa = relapath, pb = path; *pa && *pb; pa++, pb++){
                if(*pa != *pb)
                    break;
                if(*pa == '/'){
                    n--;
                    path = pb+1;
                }
            }
            while(n-- > 0)
                Bprint(o, "../");
        }
        Bprint(o, "%s\n", path);
    }
}

```

```

⟨global cleanidx 109d⟩≡ (285)
bool cleanidx = false; /* skip tree check for checkedin() */

```

```

⟨constant bdir 109e⟩≡ (285)
// -b
char *bdir = ".git/fs/HEAD/tree";

```

Uses bdir 109e.

```

⟨function checkedin 109f⟩≡ (285)
bool
checkedin(Idxent *e, bool change)
{
    char *p;
    int r;

    /* clean index, no need to check tree */

```

```

if(cleanidx)
    return e >= &idx[0] && e < &idx[nidx] && e->state != 'R';

if((p = smprint("%s/%s", bdir, e->path)) == nil)
    sysfatal("smprint: %r");

r = access(p, AEXIST);
if(r == 0 && change){
    if(e->state != 'R')
        e->state = 'T';
    staleidx = true;
}
free(p);

return r == 0;
}

```

Uses [bdir 109e](#), [cleanidx 285](#), and [staleidx](#).

<function samedata 110>≡ (285)

```

/*
 * compares whether the indexed entry 'a'
 * has the same contents and mode as
 * the entry on disk 'b'; if the indexed
 * entry is nil, does a deep comparison
 * of the checked out file and the file
 * checked in.
 */
int
samedata(Idxent *a, Idxent *b)
{
    char *gitpath, ba[IOUNIT], bb[IOUNIT];
    int fa, fb, na, nb, same;
    Dir *da, *db;

    if(a != nil){
        if(a->qid.path == b->qid.path
            && a->qid.vers == b->qid.vers
            && a->qid.type == b->qid.type
            && a->mode == b->mode
            && a->mode != 0)
            return 1;
    }

    same = 0;
    da = nil;
    db = nil;
    gitpath = smprint("%s/%s", bdir, b->path);
    if(gitpath == nil)
        sysfatal("smprint: %r");
    fa = open(gitpath, OREAD);
    fb = open(b->path, OREAD);
    if(fa == -1 || fb == -1)
        goto mismatch;
    da = dirfstat(fa);
    db = dirfstat(fb);
    if(da == nil || db == nil)
        goto mismatch;
    if((da->mode&0100) != (db->mode&0100))
        goto mismatch;
    if(da->length != db->length)

```

```

    goto mismatch;
while(true){
    if((na = readn(fa, ba, sizeof(ba))) == -1)
        goto mismatch;
    if((nb = readn(fb, bb, sizeof(bb))) == -1)
        goto mismatch;
    if(na != nb)
        goto mismatch;
    if(na == 0)
        break;
    if(memcmp(ba, bb, na) != 0)
        goto mismatch;
}
if(a != nil){
    a->qid = db->qid;
    a->mode = db->mode;
    staleidx = 1;
}
same = 1;

mismatch:
    free(da);
    free(db);
    if(fa != -1)
        close(fa);
    if(fb != -1)
        close(fb);
    return same;
}

```

Uses `bdir` 109e and `staleidx`.

9.7 Finalizations

```

⟨main() (walk.c) other locals 111a⟩+≡ (102g) <107a 112a>
    fdt wfd;
    Biobuf *w;
    Dir rn;

```

```

⟨main() (walk.c) finalizations, when isindexed && staleidx 111b⟩≡ (102g)
    wfd = create(".git/INDEX9.new", OWRITE, 0644);
    if(wfd != ERROR_NEG1){
        w = Bfdopen(wfd, OWRITE);
        if(w == nil){
            close(wfd);
            goto Nope;
        }
        for(i = 0; i < nidx; i++){
            while(i+1 < nidx && strcmp(idx[i].path, idx[i+1].path) == 0)
                i++;
            if(idx[i].state == 'U')
                continue;
            Bprint(w, "%c %Q %o %s\n",
                idx[i].state,
                idx[i].qid,
                idx[i].mode,
                idx[i].path);
        }
        Bterm(w);
    }

```

```

    nulldir(&rn);
    rn.name = "INDEX9";
    if(remove(".git/INDEX9") == -1)
        if(access(".git/INDEX9", AEXIST) == 0)
            goto Nope;
    if(dirwstat(".git/INDEX9.new", &rn) == -1)
        sysfatal("rename: %r");
}

```

`<main() (walk.c) other locals 112a>+≡ (102g) <111a 112d>`

```

char *p, *e, xbuf[8];

```

`<main() (walk.c) after Nope label 112b>≡ (102g)`

```

if(!dirty)
    exits(nil);

```

```

p = xbuf;
e = p + sizeof(xbuf);
for(i = 0; (1 << i) != Tflg; i++)
    if(dirty & (1 << i))
        p = seprint(p, e, "%c", "RMAUT"[i]);
*p = '\0';
exits(xbuf);

```

`<global dirty (walk.c) 112c>≡ (285)`

```

int dirty;

```

9.8 Advanced features

9.8.1 git/walk -b <base>

`<main() (walk.c) other locals 112d>+≡ (102g) <112a`

```

char *base;
Hash h;

```

`<main() (walk.c) command line processing 112e>+≡ (102g) <103d 112f>`

```

case 'b':
    isindexed = false;
    base = EARGF(usage());
    if(resolveref(&h, base) == ERROR_NEG1)
        sysfatal("no such ref '%s'", base);
    bdir = smprint(".git/fs/object/%H/tree", h);
    break;

```

9.8.2 git/walk -I

`<main() (walk.c) command line processing 112f>+≡ (102g) <112e 113b>`

```

case 'I':
    /* invalidate index */
    staleidx = true;
    break;

```

<main() (walk.c) initializations part 2, when stale 113a>≡ (102g)

```
nwdir = 0;
wdirsz = 32;
wdir = emalloc(wdirsz*sizeof(Idxent));

if(chdir(bdir) == -1)
    sysfatal("chdir: %r");

/* load whole tree into index when stale */
intree = 1;
if(staleidx || argc == 0)
    loadwdir(".");
else for(i = 0; i < argc; i++)
    loadwdir(argrel[i]);

if(chdir(repopath) == -1)
    sysfatal("chdir: %r");

/* use as index */
idx = wdir;
nidx = nwdir;
idxsz = wdirsz;

cleanidx = true;
```

9.8.3 git/walk -r

<main() (walk.c) command line processing 113b>+≡ (102g) <112f 114>

```
case 'r':
    findslashes(EARGF(usage()));
    break;
```

<global relapath (walk.c) 113c>≡ (285)

```
char relapath[1024];
```

<global nslash (walk.c) 113d>≡ (285)

```
int nslash;
```

<function findslashes 113e>≡ (285)

```
void
findslashes(char *path)
{
    char *p;

    p = cleannname(path);
    if(p[0] == '.') {
        if(p[1] == '\0')
            return;
        else if(p[1] == '.' && (p[2] == '/' || p[2] == '\0'))
            sysfatal("relative path escapes git root");
    }

    snprintf(relapath, sizeof relapath, "%s/", p);
    p = relapath;
    if(*p == '/')
        p++;

    for(; *p; p++)
        if(*p == '/')
```

```
        nslash++;  
    }
```

Uses `nslash` 285 and `relapath` 108b.

9.8.4 `git/walk -c`

`<main() (walk.c) command line processing 114>+≡`

`(102g) <113b`

```
    case 'c':  
        rstr = "";  
        tstr = "";  
        mstr = "";  
        astr = "";  
        ustr = "";  
        break;
```

Chapter 10

git/save

10.1 Usage

```
<function usage (git9/save.c) 115a>≡ (282b)
void
usage(void)
{
    fprintf(STDERR, "usage: %s -n name -e email -m message -d date [files...]\n", argv0);
    exits("usage");
}
```

```
<globals authorxxx(save.c) 115b>≡ (282b)
// -n
char    *authorname;
// -e
char    *authoremail;
```

```
<globals comitterxxx(save.c) 115c>≡ (282b)
// -N
char    *comittername;
// -E
char    *comitteremail;
```

```
<global commitmsg(save.c) 115d>≡ (282b)
// -m
char    *commitmsg;
```

10.2 main()

```
<function main (git9/save.c) 115e>≡ (282b)
void
main(int argc, char **argv)
{
    char cwd[1024];
    int ncwd;
    Biobuf *f; // .git/.INDEX9
    vlong date;
    errorneg1 r;
    Object *t;
    Hash th; // tree hash
    Hash ch; // commit hash
    int i;
    <main() (save.c) other locals 53a>
```

```

gitinit(nil, 0, nil);

// assumes run from project root
if(access(".git", AEXIST) != 0)
    sysfatal("could not find git repo: %r");
if(getwd(cwd, sizeof(cwd)) == nil)
    sysfatal("getcwd: %r");
ncwd = strlen(cwd);
date = time(nil);

ARGBEGIN{
<main() (save.c) command line processing 116a>
default: usage(); break;
}ARGEND;

<main() (save.c) sanity check globals after command line processing 117c>
if(committername == nil && committeremail == nil){
    committername = authorname;
    committeremail = authoremail;
}
<main() (save.c) if dstr 121b>

<main() (save.c) clean names argv 117a>
qsort(argv, argc, sizeof(*argv), namecmp);

t = findroot();

f = Bopen(".git/INDEX9", OREAD);
<main() (save.c) sanity check f 117d>
<main() (save.c) read .git/INDEX9 and set idx 53b>
qsort(idx, nidx, sizeof(Idxent), idxcmp);

r = treeify(t, argv, argv + argc, 0, &th);
<main() (save.c) sanity check r 117e>
mkcommit(&ch, date, th);

print("%H\n", ch);
exits(nil);
}

<main() (save.c) command line processing 116a>≡ (115e) 116b▷
case 'm':
    commitmsg = EARGF(usage());
    break;

<main() (save.c) command line processing 116b>+≡ (115e) <116a 116c▷
case 'n':
    authorname = EARGF(usage());
    break;
case 'e':
    authoremail = EARGF(usage());
    break;

<main() (save.c) command line processing 116c>+≡ (115e) <116b 120b▷
case 'N':
    committername = EARGF(usage());
    break;
case 'E':
    committeremail = EARGF(usage());
    break;

```

```

⟨main() (save.c) clean names argv 117a⟩≡ (115e)
for(i = 0; i < argc; i++){
    cleannname(argv[i]);
    if(*argv[i] == '/' && strcmp(argv[i], cwd, cwd) == 0)
        argv[i] += cwd;
    while(*argv[i] == '/')
        argv[i]++;
}

```

```

⟨function namecmp 117b⟩≡ (282b)
int
namecmp(void *pa, void *pb)
{
    return strcmp(*(char**)pa, *(char**)pb);
}

```

```

⟨main() (save.c) sanity check globals after command line processing 117c⟩≡ (115e)
if(commitmsg == nil)
    sysfatal("missing message");
if(authorname == nil)
    sysfatal("missing name");
if(authoremail == nil)
    sysfatal("missing email");
if((committername == nil) != (committeremail == nil))
    sysfatal("partially specified committer");

```

```

⟨main() (save.c) sanity check f 117d⟩≡ (115e)
if(f == nil)
    sysfatal("open index: %r");

```

```

⟨main() (save.c) sanity check r 117e⟩≡ (115e)
if(r == ERROR_NEG1)
    sysfatal("could not commit: %r");

```

10.3 findroot()

```

⟨function findroot 117f⟩≡ (282b)
/// main(save.c) -> <>
Object*
findroot(void)
{
    Object *t, *c;
    Hash h;

    if(resolveref(&h, "HEAD") == ERROR_NEG1)
        return emptydir();
    c = readobject(h);
    ⟨findroot() sanity check commit c 117g⟩
    t = readobject(c->commit->tree);
    ⟨findroot() sanity check tree t 117h⟩
    return t;
}

```

```

⟨findroot() sanity check commit c 117g⟩≡ (117f)
if(c == nil || c->type != GCommit)
    sysfatal("could not read HEAD %H", h);

```

```

⟨findroot() sanity check tree t 117h⟩≡ (117f)
if(t == nil)
    sysfatal("could not read tree for commit %H", h);

```

10.4 treeify()

```
<function treeify 118>≡ (282b)
  /// main(save.c) -> <>
  int
  treeify(Object *t, char **path, char **epath, int off, Hash *h)
  {
    int nent, ne, slash, isdir;
    char *s, **p, **ep;
    Dirent *e, *ent;
    Object *o;
    Dir *d;

    nent = t->tree->nent;
    ent = eamalloc(nent, sizeof(*ent));
    memcpy(ent, t->tree->ent, nent*sizeof(*ent));
    for(p = path; p != epath; p = ep){
      s = *p;

      /*
       * paths have been normalized already,
       * no leading or double-slashes allowed.
       */
      assert(off <= strlen(s));
      assert(off == 0 || s[off-1] == '/');
      assert(s[off] != '\0' && s[off] != '/');

      /* get next path element length (from off until '/' or nul) */
      for(ne = 1; s[off+ne] != '\0' && s[off+ne] != '/'; ne++)
        ;

      /* truncate at '/' or nul */
      slash = s[off + ne];
      s[off + ne] = '\0';

      /* skip over children (having s as prefix) */
      for(ep = p + 1; slash && ep != epath; ep++){
        if(strncmp(s, *ep, off + ne) != 0)
          break;
        if((*ep)[off+ne] != '\0' && (*ep)[off+ne] != '/')
          break;
      }

      d = dirstat(s);
      e = dirent(&ent, &nent, s + off);
      if(e->islink)
        sysfatal("symlinks may not be modified: %s", s);
      if(e->ismod)
        sysfatal("submodules may not be modified: %s", s);

      s[off + ne] = slash;

      isdir = d != nil && (d->mode & DMDIR) != 0;
      /*
       * exist? slash? dir?   track?
       * n      -      -      -      -> remove: file gone
       * y      n      n      y      -> blob: tracked non-dir
       * y      n      y      n      -> remove: file untracked
       * y      n      y      n      -> remove: file -> dir
       * y      n      y      y      -> remove: file -> dir
      */
    }
  }
}
```

```

* y      n      y      n      -> untracked dir, cli junk
* y      y      y      n      -> recurse
* y      y      y      y      -> recurse
*/
if(d == nil || !slash && isdir && tracked(s)){
    /*
    * if a tracked file is removed or turned
    * into a dir, we want to delete it. We
    * only want to change files passed in, and
    * not ones along the way, so ignore files
    * that have a '/'.
    */
    e->name = nil;
    s[off + ne] = slash;
    continue;
} else if(slash && isdir){
    /*
    * If we have a list of entries that go into
    * a directory, create a tree node for this
    * entry, and recurse down.
    */
    e->mode = DMDIR | 0755;
    o = readobject(e->h);
    if(o == nil || o->type != GTree)
        o = emptydir();
    /*
    * if after processing deletions, a tree is empty,
    * mark it for removal from the parent.
    *
    * Note, it is still written to the object store,
    * but this is fine -- and ensures that an empty
    * repository will continue to work.
    */
    if(treeify(o, p, ep, off + ne + 1, &e->h) == 0)
        e->name = nil;
} else if(!slash && !isdir){
    /*
    * If the file was explicitly passed in and is
    * not a dir, we want to either remove it or
    * track it, depending on the state of the index.
    */
    if(tracked(s) && !isdir)
        blobify(d, s, &e->mode, &e->h);
    else
        e->name = nil;
}
free(d);
}
if(nent == 0)
    sysfatal("%.*s: refusing to update empty directory", off, *path);
nent = writetree(ent, nent, h);
free(ent);
return nent;
}

```

Uses GCommit [33a](#), authoremail [58c](#), authorname, commitmsg [282b](#), committeremail [282b](#), committername, emptydir(), nparents [115b](#), parents, readobject() [157b](#), resolveref(), and writeobj() [56c](#).

<function dirent 119> ≡ [\(282b\)](#)
Dirent*
dirent(Dirent **ent, int *nent, char *name)

```

{
    Dirent *d;

    assert(strchr(name, '/') == nil);

    for(d = *ent; d != *ent + *nent; d++)
        if(d->name && strcmp(d->name, name) == 0)
            return d;
    // else
    *nent += 1;
    *ent = erealloc(*ent, *nent * sizeof(Dirent));
    d = *ent + (*nent - 1);
    memset(d, 0, sizeof(*d));
    d->name = estrdup(name);
    return d;
}

```

<function tracked 120a>≡ (282b)

```

bool
tracked(char *path)
{
    int r, lo, hi, mid;

    lo = 0;
    hi = nidx-1;
    while(lo <= hi){
        mid = (hi + lo) / 2;
        r = strcmp(path, idx[mid].path);
        if(r < 0)
            hi = mid-1;
        else if(r > 0)
            lo = mid+1;
        else
            return idx[mid].state != 'R';
    }
    // else
    return false;
}

```

10.5 mkcommit()

10.6 Advanced features

10.6.1 git/save -p, multi-parents (merge) commits

<main() (save.c) command line processing 120b>+≡ (115e) <116c 121a>

```

case 'p':
    if(nparents >= Maxparents)
        sysfatal("too many parents");
    if(resolveref(&parents[nparents++], EARGF(usage())) == -1)
        sysfatal("invalid parent: %r");
    break;

```

10.6.2 git/save -d, non-standard commit date

<main() (save.c) other locals 120c>+≡ (115e) <53a

```

char *dstr = nil;

```

`<main() (save.c) command line processing 121a>+≡`

`(115e) <120b`

```
case 'd':  
    dstr = EARGF(usage());  
    break;
```

`<main() (save.c) if dstr 121b>≡`

`(115e)`

```
if(dstr){  
    date=strtoll(dstr, &dstr, 10);  
    if(strlen(dstr) != 0)  
        sysfatal("could not parse date %s", dstr);  
}
```

Part II

Basic Commands

Chapter 11

Creating a Repository

11.1 Initializing a new repository: git init

11.1.1 init.rc

```
<git9/init.rc 123a>≡
#!/bin/rc -e

rfork ne
. /lib/git/common.rc
<init.rc command line processing 123c>

dir=$1
if(~ $#dir 0)
    dir=.
if(~ $#branch 0)
    branch=master
if(test -e $dir/.git)
    die $dir/.git already exists
<init.rc set $name derived from $dir 123b>
<init.rc check $upstream flag 181a>

mkdir -p $dir/.git/refs/^(heads remotes)
mkdir -p $dir/.git/^(fs objects)
>$dir/.git/config {
    echo '[core]'
    echo ' repositoryformatversion = p9.0'
    <init.rc set origin in .git/config 181b>
    echo '[branch "'$branch'"]'
    echo ' remote = origin'
}
>$dir/.git/INDEX9
>$dir/.git/HEAD {
    echo ref: refs/heads/$branch
}
exit ''

<init.rc set $name derived from $dir 123b>≡ (123a)
name='{basename '{cleanname -d '{pwd} $dir}}
```

11.1.2 Usage

```
<init.rc command line processing 123c>≡ (123a)
flagfmt='u:upstream upstream,b:branch branch'; args='name'
```

```
eval '{auxrc/getflags $*} || exec auxrc/usage
```

11.2 Copying a local repository: `cp -r`

11.3 Cloning a remote repository: `git clone`

Chapter 12

Staging a Diff

12.1 Adding files: git add

12.1.1 Usage

```
<add.rc command line processing 125a>≡ (125b)
    flagfmt='r:remove'; args='file ...'
    eval '{auxrc/getflags $*} || exec auxrc/usage
```

12.1.2 add.rc

```
<git9/add.rc 125b>≡
    #!/bin/rc -e

    rfork ne
    . /lib/git/common.rc
    # for setting $gitroot, $gitrel
    gitup
    <add.rc command line processing 125a>

    s=A

    <add.rc if $remove flag 126e>
    <add.rc sanity check $* 125c>

    <add.rc set $paths derived from $* 126b>
    walk -f ./$paths | grep -v '^(/)?\.git/' | \
        sed 's~/\'$s' NOQID 0 /' >> .git/INDEX9
    exit ''

<add.rc sanity check $* 125c>≡ (125b)
    if(~ $#* 0)
        exec auxrc/usage
```

12.1.3 gitup() part 1

```
<fn gitup(common.rc) 125d>≡ (261a)
    fn gitup{
        gitroot='${nl}{git/conf -r >[2]/dev/null}
        if(~ $#gitroot 0)
            die 'not a git repository'
        gitrel='${nl}{pwd | drop $gitroot | sed 's@~/@@'}
        if(~ $#gitrel 0)
```

```

        gitrel='.'
    if(! cd $gitroot)
        die cd $gitroot: no repo there

```

```

    <gitup() (common.rc) start git/fs 128c>

```

```

}

```

```

<global nl (common.rc) 126a>≡ (261a)
nl='
'

```

```

<add.rc set $paths derived from $* 126b>≡ (125b)
paths='${nl}{cleanname -d $gitrel $* | drop $gitroot}

```

12.1.4 drop()

```

<fn drop (common.rc) 126c>≡ (261a)
fn drop {
    awk '
    BEGIN{ARGC=0}
    {
        if(index($0, ARGV[1]) == 1)
            $0=substr($0, length(ARGV[1])+1)
        print
    }
    ' $*
}

```

12.2 Removing files: git rm

```

<git9/rm.rc 126d>≡
#!/bin/rc -e

exec git/add -r $*

```

```

<add.rc if $remove flag 126e>≡ (125b)
if(~ $remove 1)
    s=R

```

Chapter 13

Committing a Diff

13.1 Committing the index: git commit

13.1.1 Usage

```
<commit.rc command line processing 127a>≡ (127b)
  flagfmt='m:msg message, r:revise, e:edit, p:partial'; args='[file ...]'
  eval '{auxrc/getflags $*} || exec auxrc/usage
```

13.1.2 commit.rc

```
<git9/commit.rc 127b>≡
  #!/bin/rc -e

  rfork ne
  . /lib/git/common.rc

  <fn findbranch(commit.rc) 128d>
  <fn cleanmsg(commit.rc) 130a>
  <fn editmsg(commit.rc) 129e>
  <fn parents(commit.rc) 129b>
  <fn commit(commit.rc) 130c>
  <fn update(commit.rc) 130d>
  <fn filterdiff(commit.rc) 131e>
  <fn pickhunks(commit.rc) 131d>
  <fn apply(commit.rc) 132>

  gitup

  <commit.rc command line processing 127a>

  <commit.rc create $msgfile.tmp 129d>

  files=()
  <commit.rc adjust $files if merge parents 139a>
  if(! ~ $#* 0)
    files=($files '$nl{git/walk -fRMA -c '$nl{cleanname -d $gitrel $*}})
  if(~ $status '' || ~ $#files 0 && ! test -f .git/merge-parents && ~ $#revise 0)
    die 'nothing to commit'
  <commit.rc if $partial 131c>

  @{
    flag e +
```

```

# prepare the commit
whoami
findbranch
parents
editmsg

commit
update
} || die 'could not commit:' $status
exit ''

```

13.1.3 whoami()

`<fn whoami(common.rc) 128a>` ≡ (261a)

```

fn whoami{
    name='${git/conf user.name}'
    email='${git/conf user.email}'
    <whomai() (common.rc) if adm keys 128b>
    if(~ $name '')
        name=$user
    if(~ $email '')
        email=$user@$sysname
    status=''
}

```

`<whomai() (common.rc) if adm keys 128b>` ≡ (128a)

```

if(test -f /adm/keys.who){
    if(~ $name '')
        name='${awk -F'|' ' $1=="$user" {x=$3} END{print x}' </adm/keys.who}'
    if(~ $email '')
        email='${awk -F'|' ' $1=="$user" {x=$5} END{print x}' </adm/keys.who}'
}

```

13.1.4 gitup() part 2

`<gitup() (common.rc) start git/fs 128c>` ≡ (125d)

```

gitfs=$gitroot/.git/fs
startfs=()
mkdir -p $gitfs
if(! test -e $gitfs/ctl)
    startfs=true
if(! grep -s '^repo '$gitroot'$' $gitfs/ctl >[2]/dev/null)
    startfs=true
if(~ $#startfs 1)
    git/fs >[1=2]
if not
    status=''

```

13.1.5 findbranch()

`<fn findbranch(commit.rc) 128d>` ≡ (127b)

```

fn findbranch{
    branch='{git/branch}'
    <findbranch() (commit.rc) set refilepath and initial 129a>
}

```

```

<findbranch() (commit.rc) set refpath and initial 129a>≡ (128d)
if(test -e $gitfs/branch/$branch/tree){
    refpath=.git/refs/$branch
    initial=false
}
if not if(test -e $gitfs/object/$branch/tree){
    refpath=.git/HEAD
    initial=false
}
if not if(! test -e $gitfs/HEAD/tree){
    refpath=.git/refs/$branch
    initial=true
}
if not
    die 'invalid branch:' $branch

```

13.1.6 parents()

```

<fn parents (commit.rc) 129b>≡ (127b)
fn parents{
    <parents() (commit.rc) if revise 131a>
    <parents() (commit.rc) if merge parents 139b>
    <parents() (commit.rc) if initial 129c>
    if not
        parents='{git/query $branch}'
}

```

```

<parents() (commit.rc) if initial 129c>≡ (129b)
if not if(~ $initial true)
    parents=()

```

13.1.7 editmsg()

```

<commit.rc create $msgfile.tmp 129d>≡ (127b)
wdir=/mnt/git
ramfs -m $wdir
msgfile=$wdir/git-msg.$pid
if(~ $#msg 1)
    echo $msg >$msgfile.tmp
<commit.rc when create $msgfile.tmp, else if revise 131b>

```

```

<fn editmsg (commit.rc) 129e>≡ (127b)
fn editmsg{
    if(! test -s $msgfile.tmp){
        >$msgfile.tmp {
            echo '# Author:' $name '<'$email'>'
            echo '#'
            for(p in $parents)
                echo '# parent:' $p
            git/walk -fAMR $files | subst '~' '# '
            echo '#'
            echo '# Commit message:'
        }
        edit=1
    }
    if(! ~ $#edit 0){
        giteditor='{git/conf core.editor}'
        if(~ $#editor 0)

```

```

        editor=$giteditor
        if(~ $#editor 0)
            editor=hold
        $editor $msgfile.tmp
    }
    cleanmsg < $msgfile.tmp > $msgfile
    if(! test -s $msgfile)
        die 'empty commit message'
}

```

`<fn cleanmsg(commit.rc) 130a>≡` (127b)

```

# Remove commentary lines.
# Remove leading and trailing empty lines.
# Combine consecutive empty lines between paragraphs.
# Remove trailing spaces from lines.
# Ensure there's trailing newline.
fn cleanmsg{
    awk '
    /^[ \t]*#/ {next}
    /^[ \t]*$/ {empty = 1; next}

    wet && empty {printf "\n"}
    {wet = 1; empty = 0}
    {sub(/[\t]+$/, ""); print $0}
    '
}

```

`<fn subst(common.rc) 130b>≡` (261a)

```

fn subst {
    awk '
    BEGIN{ARGC=0}
    {sub(ARGV[1], ARGV[2]); print}
    ' $*
}

```

13.1.8 commit()

`<fn commit(commit.rc) 130c>≡` (127b)

```

fn commit{
    msg='' {cat $msgfile}
    if(! ~ $#parents 0)
        pflags='-p'^$parents
    hash='{git/save -n "$name -e "$email -m "$msg $pflags $files || die $status}
}

```

13.1.9 update()

`<fn update(commit.rc) 130d>≡` (127b)

```

fn update{
    mkdir -p '{basename -d $refpath}
    # Paranoia: let's not mangle the repo.
    if(~ $#hash 0)
        die 'botched commit'
    rm -f .git/merge-parents
    echo $branch: $hash
    echo $hash > $refpath
    for(f in $files){
        if(! test -e $f && ! test -e .git/object/$hash/tree/$f)

```

```

        echo R NOQID 0 $f >> .git/INDEX9
    if not
        echo T NOQID 0 $f >> .git/INDEX9
}
}

```

13.2 git/walk -fRMA

13.3 git/query \$branch

13.4 git/save

13.5 Advanced features

13.5.1 git/commit -r, revising a commit

```

<parents() (commit.rc) if revise 131a>≡ (129b)
    if(! ~ $#revise 0)
        parents='{cat $gitfs/HEAD/parent}

```

```

<commit.rc when create $msgfile.tmp, else if revise 131b>≡ (129d)
    if not if(~ $#revise 1){
        msg=1
        echo revising commit '{cat $gitfs/HEAD/hash}
        cat $gitfs/HEAD/msg >$msgfile.tmp
    }

```

13.5.2 git/commit -p, partial commit and hunks selection

```

<commit.rc if $partial 131c>≡ (127b)
    if(~ $#partial 1){
        for(i in $files)
            pickhunks $i
        apply
    }

```

```

<fn pickhunks(commit.rc) 131d>≡ (127b)
    fn pickhunks{
        p=$wdir/patch
        if(test -f .git/fs/HEAD/tree/$i)
            #<[12]/fd/0 >[13]/fd/1 { diff -u .git/fs/HEAD/tree/$i $i | filterdiff >>$p }
            { diff -u .git/fs/HEAD/tree/$i $i | filterdiff >>$p } || status=''
        if not
            diff -u /dev/null $i >>$p || status=''
    }

```

```

<fn filterdiff(commit.rc) 131e>≡ (127b)
    fn filterdiff{
        awk '
            function dump(){
                if(n == 0)
                    return
                print "--- a/"f >>"/dev/cons"
                print "+++ b/"f >>"/dev/cons"
            }

```

```

        for(i = 0; i < n; i++)
            print lines[i] >>"/dev/cons"
        printf("Commit this hunk?[n] ") >>"/dev/cons"
        getline ans < "/dev/cons"
        if(ans ~ /[yY]/)
            for(i = 0; i < n; i++)
                print lines[i]
        n = 0
    }
    /~---/{ next }
    /~\+\+\+\+/{
        f = substr($0, 5)
        print "--- "f
        print "+++ "f
        n = 0
        next
    }
    /~@@/{ dump() }
    { lines[n++] = $0 }
    END{ dump() }
,
}

```

`<fn apply(commit.rc) 132>≡ (127b)`

```

fn apply{
    t=$wdir/tree/
    p=$wdir/patch
    ln='{wc -l $p}'
    if(test $ln(1) -lt 3)
        die 'no hunks seleted'
    f='{patch -n $p}'
    mkdir $t
    @{cd .git/fs/HEAD/tree; tar c $f >[2]/dev/null} | @{cd $t; tar xT}
    for(i in $f){
        d='{basename -d $i}'
        if(! ~ $d .){
            mkdir -p $t^$d
            bind -bc $t^$d $d
        }
    }
    bind -bc $t .
    patch <$p >/dev/null || die 'failed partial patch: '^$status
}

```

Chapter 14

Branching

14.1 git branch

14.1.1 Usage

```
<branch.rc command line processing 133a>≡ (133b)
  flagfmt='a:listall, b:baseref ref, r:remove, n:newbr, s:stay, m:merge, M:nomod'
  args=' [branch] '
  eval '{auxrc/getflags $*} || exec auxrc/usage
```

14.1.2 branch.rc

```
<git9/branch.rc 133b>≡
  #!/bin/rc -e

  rfork en
  . /lib/git/common.rc
  gitup
  <branch.rc command line processing 133a>
  <branch.rc globals 134b>
  <branch.rc when no args 133c>
  // else
  if(! ~ $#* 1)
    exec auxrc/usage
  // else
  branch=$1
  if(~ $branch refs/heads/*)
    new=$name
  if not if(~ $branch heads/*)
    new=refs/$branch
  if not
    new=refs/heads/$branch

  # where are we now?
  orig='{git/query HEAD}
  origbranch=refs/{awk ' $1=="branch"{print $2}' < $gitfs/ctl}

  <branch.rc when single branch arg, new, orig, origbranch set 134e>
```

14.2 Listing branches: git branch

```
<branch.rc when no args 133c>≡ (133b)
```

```

if(~ $#* 0){
    if(~ $#listall 0)
        awk ' $1=="branch"{print $2}' < $gitfs/ctl
    if not
        cd .git/refs/ && walk -f heads remotes | sort
    exit
}

```

14.3 Creating a branch: git branch <name>

```

<branch.rc adjust baseref if existing remote branch 134a>≡ (134e)
# if we're switching branches to a branch that doesn't exist,
# but we have a remote branch that exists upstream, create a
# new head to mirror it.
if(~ $#newbr 0){
    if(! ~ $#baseref 0)
        die update would clobber $branch with $baseref
    if(! test -e .git/$new){
        baseref='${nl{echo -n $new | sed s@refs/heads/@refs/remotes/origin/@}
        if(! base='{git/query $baseref})
            exit 'bad ref'
    }
}

```

```

<branch.rc globals 134b>≡ (133b)
modified=()
deleted=()
failed=()

```

```

<branch.rc set modified and deleted 134c>≡ (134e)
modified='${nl{git/query -c HEAD $base | grep '^[-]' | subst '^..'}'
deleted='${nl{git/query -c HEAD $base | grep '^-' | subst '^..'}'

```

```

<branch.rc sanity checks when remove or when modified or deleted 134d>≡ (134e)
# if we remove the current branch without switching, bad things happen
if(~ $remove 1 && ~ $origbranch $new)
    die 'cannot remove current branch'
# if we're not merging, don't clobber existing changes.
if(~ $#merge 0 && ~ $#remove 0){
    if(! ~ $#modified 0 || ! ~ $#deleted 0){
        git/walk -fRMA $modified $deleted ||
            die 'uncommitted changes would be clobbered'
    }
}

```

```

<branch.rc when single branch arg, new, orig, origbranch set 134e>≡ (133b)

```

```

<branch.rc adjust baseref if existing remote branch 134a>
# figure out where we want the new branch to point to
if (~ $#baseref 1)
    base='{git/query $baseref} || exit 'bad ref'
if not if(~ $#newbr 0)
    base='{git/query $new} || exit 'bad ref'
if not
    base='{git/query HEAD} || exit 'bad ref'

```

```

<branch.rc set modified and deleted 134c>

```

```

⟨branch.rc sanity checks when remove or when modified or deleted 134d⟩
⟨branch.rc if remove 136⟩
// else
commit='{git/query $base} || die 'branch does not exist:' $base
basedir='{git/query -p $base}
dirtypaths=()
if(! ~ $#modified 0 || ! ~ $#deleted 0)
    dirtypaths='${nl}{git/walk -cfrMA $modified $deleted}

>>[3] .git/INDEX9{
    for(d in $deleted){
        if(! test -d $d){
            rm -f $d
            echo R NOQID 0 $d >>[1=3]
        }
    }
    for(m in $modified $deleted) if(! ~ $m $dirtypaths){
        d='${nl}{basename -d $m}
        mkdir -p $d
        # Modifications can turn a file into
        # a directory, or vice versa, so we
        # need to delete and copy the files
        # over.
        a=dir
        b=dir
        if(test -f $m)
            a=file
        if(test -f $basedir/tree/$m)
            b=file
        if(! ~ $a $b){
            rm -rf $m
            echo R NOQID 0 $m >>[1=3]
        }
        if(~ $b file){
            echo T NOQID 0 $m >>[1=3]
            cp -x -- $basedir/tree/$m $m || {echo cp failed: $m; failed=1}
            touch -c $m
        }
    }
}
modified=()
deleted=()

for(ours in $dirtypaths){
    common=$gitfs/object/$orig/tree/$ours
    theirs=$gitfs/object/$base/tree/$ours
    merge1 $ours $ours $common $theirs
    st=$status
    if(! ~ $st '')
        >[1=2] echo merge failed $ours: $st
}
dirtypaths=()

if(~ $new */)
    mkdir -p .git/'{basename -d $new}
echo $commit > .git/$new
if(! ~ $#stay 0)
    exit
if(~ $failed 1){
    echo 'pull failed: fix errors and try again' >[1=2]
}

```

```
        exit nopull
    }
    echo ref: $new > .git/HEAD
    echo $new: '{git/query $new}'
    exit ''
```

14.4 Deleting a branch: `git branch -d <name>`

```
<branch.rc if remove 136>≡ (134e)
    if(~ $remove 1){
        rm -f .git/$new
        echo 'removed branch' $new
        exit
    }
```

14.5 Checking out a branch: `git checkout`

14.5.1 Computing the index (and work tree) from trees

14.5.2 Creating a file from a blob

14.6 Resetting a branch: `git reset`

Chapter 15

Merging Branches

15.1 merging branches: git merge

15.1.1 Usage

```
<merge.rc command line processing 137a>≡ (137b)
  flagfmt=''; args='theirs'
  eval '{auxrc/getflags $*} || exec auxrc/usage
```

15.1.2 merge.rc

```
<git9/merge.rc 137b>≡
#!/bin/rc -e

rfork ne
. /lib/git/common.rc
<fn merge(merge.rc) 138a>
gitup
<merge.rc command line processing 137a>

if(! ~ $#* 1)
  exec auxrc/usage

theirs='{git/query $1}
ours='{git/query HEAD}
base='{git/query $theirs ^ ' ' ^ $ours ^ '@}'

if(~ $base $theirs)
  die 'nothing to merge, doofus'
if(! git/walk -q)
  die 'dirty work tree, refusing to merge'
if(~ $base $ours){
  >[1=2] echo 'fast forwarding...'
  echo $theirs > .git/refs/{git/branch}
  git/revert .
  exit ''
}
echo $ours >> .git/merge-parents
echo $theirs >> .git/merge-parents

merge $ours $base $theirs
echo 'merge complete: remember to commit'
exit ''
```

15.1.3 merge()

```
<fn merge(merge.rc) 138a>≡ (137b)
fn merge{
    ourbr=$gitfs/object/$1/tree
    basebr=$gitfs/object/$2/tree
    theirbr=$gitfs/object/$3/tree

    all='${nl}{git/query -c $1 $2; git/query -c $2 $3} | sed 's/^..//' | sort | uniq}
    for(f in $all){
        ours=$ourbr/$f
        base=$basebr/$f
        theirs=$theirbr/$f
        merge1 ./ $f $ours $base $theirs
    }
}
```

15.1.4 merge1()

```
<fn merge1(common.rc) 138b>≡ (261a)
# merge1 out ours base theirs
fn merge1 {@{
    rfork e
    n=$pid
    out=$1
    ours=$2
    base=$3
    theirs=$4
    tmp=$out.tmp
    while(test -f $tmp){
        tmp=$tmp.$n
        n='{echo $n + 1 | hoc}
    }

    if(! test -f $ours)
        ours=/dev/null
    if(! test -f $base)
        base=/dev/null
    if(! test -f $theirs)
        theirs=/dev/null

    if(mergeperm $ours $base $theirs){
        mkdir -p '${nl}{basename -d $tmp}
        if(! merge3 $ours $base $theirs > $tmp)
            echo merge needed: $out >[1=2]
        mv $tmp $out
        git/add $out
        chmod $mergedperms $out
    }
    if not {
        rm -f $tmp $out
        echo R NOQID 0 $out >> $gitroot/.git/INDEX9
    }
}}
```

15.1.5 mergeperm()

```
<fn mergeperm(common.rc) 138c>≡ (261a)
```

```

fn mergeperm {
    if(~ $1 /dev/null && cmp $2 $3>/dev/null)
        status=gone
    if not if (~ $3 /dev/null && cmp $1 $2>/dev/null)
        status=gone
    if not {
        mergedperms='-x'
        if(test -x $2){
            if(test -x $1 -a -x $3)
                mergedperms='+x'
        }
        if not{
            if(test -x $1 -o -x $3)
                mergedperms='+x'
        }
        status=()
    }
}

```

15.2 Merging two trees

15.3 Merging two files

15.4 diff3 and merge3

15.5 Merging conflicts

15.6 Committing merged branches: git commit

```

<commit.rc adjust $files if merge parents 139a>≡ (127b)
    if(test -f .git/merge-parents)
        files='${nl}{git/query -c '{cat .git/merge-parents} | sed 's/^...//'}

```

```

<parents()(commit.rc) if merge parents 139b>≡ (129b)
    if not if(test -f .git/merge-parents)
        parents='{cat .git/merge-parents | sort | uniq}

```

Chapter 16

Inspecting

16.1 Showing the commit history: git log

16.1.1 Usage

```
<function usage (git9/log.c) 140a>≡ (272)
static void
usage(void)
{
    fprintf(STDERR, "usage: %s [-s] [-e expr | -c commit] files..\n", argv0);
    exits("usage");
}
```

```
<global commitid(log.c) 140b>≡ (272)
// -c
// option<ref_ownd<string>> (if None then defaults to "HEAD")
char *commitid;
```

16.1.2 main()

```
<global out(log.c) 140c>≡ (272)
Biobuf *out;
```

```
<function main (git9/log.c) 140d>≡ (272)
void
main(int argc, char **argv)
{
    char repo[1024];
    int nrel, nrepo;
    <main() (log.c) other locals 143c>

    ARGBEGIN{
    <main() (log.c) command line processing 141a>
    default: usage(); break;
    }ARGEND;

    gitinit(repo, sizeof(repo), &nrel);
    nrepo = strlen(repo);

    <main() (log.c) if argc 143d>
    // !!chdir!!
    if(chdir(repo) == ERROR_NEG1)
        sysfatal("chdir: %r");
```

```

out = Bfdopen(STDOUT, OWRITE);

⟨main() (log.c) if queryexpr 146c⟩
else
    showcommits(commitid);

Bterm(out);
exits(nil);
}

```

```

⟨main() (log.c) command line processing 141a⟩≡ (140d) 141c▷
case 'c':
    commitid = EARGF(usage());
    break;

```

```

⟨global msgcount (log.c) 141b⟩≡ (272)
int msgcount = -1; // infinite

```

```

⟨main() (log.c) command line processing 141c⟩+≡ (140d) <141a 145c▷
case 'n':
    msgcount = atoi(EARGF(usage()));
    break;

```

16.1.3 showcommits()

```

⟨global objq (log.c) 141d⟩≡ (272)
Objq objq;

```

```

⟨global done (log.c) 141e⟩≡ (272)
Objset done;

```

```

⟨function showcommits 141f⟩≡ (272)
/// main(log.c) -> <>
static void
showcommits(char *c)
{
    Object *o, *p;
    Qelt e;
    int i;
    Hash h;

    if(c == nil)
        c = "HEAD";
    if(resolveref(&h, c) == -1)
        sysfatal("resolve %s: %r", c);
    o = readobject(h);
    ⟨showcommits() sanity check o is a valid commit object 142a⟩

    qinit(&objq);
    osinit(&done);
    qput(&objq, o, 0);

    while(qpop(&objq, &e) && (msgcount == -1 || msgcount > 0)){
        if(matchesfilter(e.o)){
            show(e.o);
            if(msgcount != -1)
                msgcount--;
        }
        for(i = 0; i < e.o->commit->nparent; i++){

```

```

        if(oshas(&done, e.o->commit->parent[i]))
            continue;
        // else
        p = readobject(e.o->commit->parent[i]);
        if(p == nil)
            sysfatal("load %H: %r", o->commit->parent[i]);
        osadd(&done, p);
        qput(&objq, p, 0);
    }
    unref(e.o);
}
}

```

Uses objq, osadd() 37e, oshas() 38d, qput() 40a, readobject() 157b, and unref() 34a.

```

⟨showcommits() sanity check o is a valid commit object 142a⟩≡ (141f)
    if(o == nil)
        sysfatal("load %H: %r", h);
    if(o->type != GCommit)
        sysfatal("%s: not a commit", c);

```

16.1.4 show()

```

⟨function show 142b⟩≡ (272)
    /// showcommits -> <>
    static void
    show(Object *o)
    {
        char *p, *q, *e;

        assert(o->type == GCommit);
        ⟨show() if shortlog 145d⟩
        else{
            Bprint(out, "Hash:\t%H\n", o->hash);
            Bprint(out, "Author:\t%s\n", o->commit->author);
            if(o->commit->committer != nil
                && strcmp(o->commit->author, o->commit->committer) != 0)
                Bprint(out, "Committer:\t%s\n", o->commit->committer);
            Bprint(out, "Date:\t%s", ctime(o->commit->mtime));
            Bprint(out, "\n");
            p = o->commit->msg;
            e = p + o->commit->nmsg;
            for(; p != e; p = q){
                q = nextline(p, e);
                Bputc(out, '\t');
                Bwrite(out, p, q - p);
                Bputc(out, '\n');
                if(q != e)
                    q++;
            }
            Bprint(out, "\n");
        }
        Bflush(out);
        return;
    }
}

```

Uses matchesfilter() 144b, msgcount, out 143a, readobject() 157b, resolverefs(), and unref() 34a.

16.1.5 Advanced features

git log <files>, path filtering

```
<struct Pfilt 143a>≡ (272)
struct Pfilt {
    char    *elt;
    bool show;

    Pfilt   *sub;
    int nsub;

};
```

```
<global pathfilt(log.c) 143b>≡ (272)
// option<ref_own<Pfilt>>
Pfilt *pathfilt;
```

```
<main()(log.c) other locals 143c>≡ (140d)
char path[1024];
char *p;
char *r;
int i;
```

```
<main()(log.c) if argc 143d>≡ (140d)
if(argc != 0){
    if(getwd(path, sizeof(path)) == nil)
        sysfatal("getwd: %r");
    if(strncmp(path, repo, nrepo) != 0)
        sysfatal("path shifted??");

    p = path + nrepo;
    pathfilt = emalloc(sizeof(Pfilt));
    for(i = 0; i < argc; i++){
        if(*argv[i] == '/'){
            if(strncmp(argv[i], repo, nrepo) != 0)
                continue;
            r = smprint("./%s", argv[i]+nrepo);
        }else
            r = smprint("./%s/%s", p, argv[i]);
        cleannamename(r);
        filteradd(pathfilt, r);
        free(r);
    }
}
```

```
<function filteradd 143e>≡ (272)
void
filteradd(Pfilt *pf, char *path)
{
    char *p, *e;
    int i;

    if((e = strchr(path, '/')) != nil)
        p = smprint("%.*s", (int)(e - path), path);
    else
        p = strdup(path);

    while(e != nil && *e == '/')
        e++;
    for(i = 0; i < pf->nsub; i++){
```

```

    if(strcmp(pf->sub[i].elt, p) == 0){
        pf->sub[i].show = pf->sub[i].show || (e == nil);
        if(e != nil)
            filteradd(&pf->sub[i], e);
        free(p);
        return;
    }
}
pf->sub = earealloc(pf->sub, pf->nsub+1, sizeof(Pfilt));

pf->sub[pf->nsub].elt = p;
pf->sub[pf->nsub].show = (e == nil);
pf->sub[pf->nsub].nsub = 0;
pf->sub[pf->nsub].sub = nil;
if(e != nil)
    filteradd(&pf->sub[pf->nsub], e);
pf->nsub++;
}

```

Uses Zhash 30b.

<function matchesfilter 144a>≡ (272)

```

bool
matchesfilter(Object *o)
{
    Object *t, *p, *pt;
    int i;
    bool r;

    assert(o->type == GCommit);
    if(pathfilt == nil)
        return true;
    t = readobject(o->commit->tree);
    if(t == nil)
        sysfatal("read %H: %r", o->commit->tree);
    for(i = 0; i < o->commit->nparent; i++){
        p = readobject(o->commit->parent[i]);
        if(p == nil)
            sysfatal("read %H: %r", o->commit->parent[i]);
        pt = readobject(p->commit->tree);
        if(pt == nil)
            sysfatal("read %H: %r", o->commit->tree);
        r = matchesfilter1(pathfilt, t, pt);
        unref(p);
        unref(pt);
        if(r)
            return true;
    }
    if(o->commit->nparent == 0)
        return matchesfilter1(pathfilt, t, nil);
    return false;
}

```

Uses GCommit 33a, matchesfilter1() 143e, pathfilt, and shortlog 140c.

<function matchesfilter1 144b>≡ (272)

```

bool
matchesfilter1(Pfilt *pf, Object *t, Object *pt)
{
    Object *a, *b;
    Hash ha, hb;
    int i, r;
}

```

```

if(pf->show)
    return true;
if(t != nil){
    if(pt != nil && t->type != pt->type)
        return true;
    if(t->type != GTree)
        return false;
}

for(i = 0; i < pf->nsub; i++){
    ha = lookup(&pf->sub[i], t);
    hb = lookup(&pf->sub[i], pt);
    if(hasheq(&ha, &hb))
        continue;
    a = readobject(ha);
    b = readobject(hb);
    r = matchesfilter1(&pf->sub[i], a, b);
    unref(a);
    unref(b);
    if(r)
        return true;
}
return false;
}

```

Uses GCommit 33a, matchesfilter1() 143e, pathfilt, readobject() 157b, and unref() 34a.

```

<function lookup (git9/log.c) 145a>≡ (272)
Hash
lookup(Pfilt *pf, Object *o)
{
    int i;

    if(o == nil)
        return Zhash;
    for(i = 0; i < o->tree->nent; i++)
        if(strcmp(o->tree->ent[i].name, pf->elt) == 0)
            return o->tree->ent[i].h;
    return Zhash;
}

```

Uses GTree 33a and hasheq().

git log -s, short log

```

<global shortlog(log.c) 145b>≡ (272)
bool shortlog;

```

```

<main() (log.c) command line processing 145c>+≡ (140d) <141c 146b>
case 's':
    shortlog=true;
    break;

```

```

<show() if shortlog 145d>≡ (142b)
if(shortlog){
    p = o->commit->msg;
    e = p + o->commit->nmsg;
    q = nextline(p, e);
    Bprint(out, "%H ", o->hash);
    Bwrite(out, p, q - p);
}

```

```

    Bputc(out, '\n');
}

```

Uses out [143a](#).

git log -e, commit expression

```

⟨global queryexpr(log.c) 146a⟩≡ (272)
char *queryexpr;

```

```

⟨main()(log.c) command line processing 146b⟩+≡ (140d) <145c
case 'e':
    queryexpr = EARGF(usage());
    break;

```

```

⟨main()(log.c) if queryexpr 146c⟩≡ (140d)
if(queryexpr != nil)
    showquery(queryexpr);

```

```

⟨function showquery 146d⟩≡ (272)
static void
showquery(char *q)
{
    Object *o;
    Hash *h;
    int n, i;

    n = resolverefs(&h, q);
    if(n == -1)
        sysfatal("resolve: %r");

    for(i = 0; i < n && (msgcount == -1 || msgcount > 0); i++){
        o = readobject(h[i]);
        if(o == nil)
            sysfatal("read %H: %r", h[i]);
        if(matchesfilter(o)){
            show(o);
            if(msgcount != -1)
                msgcount--;
        }
        unref(o);
    }
    exits(nil);
}

```

Uses GCommit [33a](#), matchesfilter() [144b](#), msgcount, objq, osinit() [37c](#), qinit() [39d](#), qpop(), qput() [40a](#), readobject() [157b](#), and resolveref().

16.2 Representing changes

16.2.1 Tree changes

16.2.2 File changes

16.3 Showing file differences: git diff

16.3.1 Usage

```

⟨diff.rc command line processing 146e⟩≡ (147)

```

```
flagfmt='c:commit branch, s:summarize, u:uncommitted'; args='[file ...]'  
eval '{auxrc/getflags $*} || exec auxrc/usage
```

16.3.2 diff.rc

```
<git9/diff.rc 147>≡  
#!/bin/rc  
  
rfork ne  
./lib/git/common.rc  
gitup  
<diff.rc command line processing 146e>  
  
bparam=(-b $commit)  
if(~ $#commit 0){  
    commit=HEAD  
    cparam=()  
    bparam=()  
}  
  
files=()  
filt=MAR  
if(~ $#uncommitted 1)  
    filt=MARU  
if(! ~ $#* 0)  
    files='{cleanname -d $gitrel $*}'  
  
branch='{git/query -p $commit}'  
if(~ $summarize 1 || ~ $uncommitted 1){  
    git/walk $bparam -r$gitrel -f$filt $cparam $files  
    exit  
}  
  
showed=()  
mntgen /mnt/scratch  
bind $branch/tree/ /mnt/scratch/a  
bind . /mnt/scratch/b  
for(f in '$nl{git/walk $bparam -c -f$filt $cparam $files}){  
    if(~ $#showed 0){  
        echo diff '{git/query $commit} uncommitted  
        showed=1  
    }  
    cd /mnt/scratch  
    a=a/$f  
    b=b/$f  
    if(! test -f a/$f)  
        a=/dev/null  
    if(! test -f b/$f)  
        b=/dev/null  
    diff -u $a $b  
}  
exit ''
```

16.3.3 `diff -u`

16.3.4 Computing tree changes

16.3.5 Computing file changes

16.3.6 Showing changes

Chapter 17

Packing and Unpacking

17.1 Packing objects: git repack

17.1.1 Usage

```
<function usage (git9/repack.c) 149a>≡ (282a)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-d]\n", argv0);
    exits("usage");
}
```

17.1.2 main.c

```
<function main (git9/repack.c) 149b>≡ (282a)
void
main(int argc, char **argv)
{
    fdt fd;
    // growing_array<Hash> (len = nrefs)
    Hash *refs = nil;
    int nrefs;
    char **names;
    Hash h;
    Dir rn;
    char path[128];

    ARGBEGIN{
        <main() (repack.c) command line processing 254c>
        default: usage();
    }ARGEND;

    gitinit(nil, 0, nil);
    nrefs = listrefs(&refs, &names);
    <main() (repack.c) sanity check nrefs 150b>
    fd = create(TMPPATH("pack.tmp"), OWRITE, 0644);
    <main() (repack.c) sanity check fd 150c>

    if(writepack(fd, refs, nrefs, nil, 0, &h) == -1)
        sysfatal("writepack: %r");
    if(indexpack(TMPPATH("pack.tmp"), TMPPATH("idx.tmp"), h) == ERROR_NEG1)
        sysfatal("indexpack: %r");
    close(fd);
}
```

```

    nulldir(&rn);
    rn.name = path;
    snprintf(path, sizeof(path), "%H.pack", h);
    if(dirwstat(TMPPATH("pack.tmp"), &rn) == ERROR_NEG1)
        sysfatal("rename pack: %r");
    snprintf(path, sizeof(path), "%H.idx", h);
    if(dirwstat(TMPPATH("idx.tmp"), &rn) == ERROR_NEG1)
        sysfatal("rename pack: %r");
    if(cleanup(h) == ERROR_NEG1)
        sysfatal("cleanup: %r");
    exits(nil);
}

```

```

⟨macro TMPPATH 150a⟩≡ (282a)
#define TMPPATH(suff) (".git/objects/pack/repack."suff)

```

```

⟨main() (repack.c) sanity check nrefs 150b⟩≡ (149b)
if(nrefs == -1)
    sysfatal("load refs: %r");

```

```

⟨main() (repack.c) sanity check fd 150c⟩≡ (149b)
if(fd == -1)
    sysfatal("open %s: %r", TMPPATH("pack.tmp"));

```

17.1.3 listrefs()

```

⟨function listrefs 150d⟩≡ (281)
/// main(repack.c) -> <>
int
listrefs(Hash **refs, char ***names)
{
    int nrefs;

    *refs = nil;
    *names = nil;
    nrefs = 0;
    if(readrefdir(refs, names, &nrefs, ".git/refs", "") == ERROR_NEG1){
        free(*refs);
        return ERROR_NEG1;
    }
    return nrefs;
}

```

Uses readrefdir() 281.

```

⟨function readrefdir 150e⟩≡ (281)
errorneg1
readrefdir(Hash **refs, char ***names, int *nrefs, char *dpath, char *dname)
{
    Dir *d, *e, *dir;
    char *path, *name, *sep;
    int ndir;

    ndir = slurpdir(dpath, &dir);
    if(ndir == -1)
        return ERROR_NEG1;
    sep = (*dname == '\\0') ? "" : "/";
    e = dir + ndir;
    for(d = dir; d != e; d++){

```

```

path = smprint("%s/%s", dpath, d->name);
name = smprint("%s%s%s", dname, sep, d->name);
if(d->mode & DMDIR) {
    // recurse
    if(readreaddir(refs, names, nrefs, path, name) == ERROR_NEG1)
        goto noref;
}else{
    *refs = erealloc(*refs, (*nrefs + 1)*sizeof(Hash));
    *names = erealloc(*names, (*nrefs + 1)*sizeof(char*));
    if(resolveref(&(*refs)[*nrefs], name) == ERROR_NEG1)
        goto noref;
    (*names)[*nrefs] = name;
    *nrefs += 1;
    goto next;
}
noref:
    free(name);
next:
    free(path);
}
free(dir);
return OK_0;
}

```

Uses `readreaddir()` 281, `resolveref()`, and `slurpdir()` 260.

17.1.4 cleanup()

```

<function cleanup 151>≡ (282a)
int
cleanup(Hash h)
{
    char newpfx[42], dpath[256], fpath[256];
    int i, j, nd;
    Dir *d;

    snprintf(newpfx, sizeof(newpfx), "%H.", h);
    for(i = 0; i < 256; i++){
        snprintf(dpath, sizeof(dpath), ".git/objects/%02x", i);
        if((nd = slurpdir(dpath, &d)) == -1)
            continue;
        for(j = 0; j < nd; j++){
            snprintf(fpath, sizeof(fpath), ".git/objects/%02x/%s", i, d[j].name);
            remove(fpath);
        }
        remove(dpath);
        free(d);
    }
    snprintf(dpath, sizeof(dpath), ".git/objects/pack");
    if((nd = slurpdir(dpath, &d)) == -1)
        return -1;
    for(i = 0; i < nd; i++){
        if(strncmp(d[i].name, newpfx, strlen(newpfx)) == 0)
            continue;
        snprintf(fpath, sizeof(fpath), ".git/objects/pack/%s", d[i].name);
        remove(fpath);
    }
    return 0;
}

```

Uses `slurpdir()` 260.

17.2 Data structures

17.2.1 Packf and packf

```
<struct Packf 152a>≡ (277)
struct Packf {
    // .git/objects/pack/xxx.pack
    char path[128];
    // option<Biobuf>, opened .git/objects/pack/xxx.pack when Some
    Biobuf *pack;

    <Pack idx fields 152c>
    <Pack extra fields 160c>
};
```

```
<globals packf 152b>≡ (277)
// array<ref_own<Packf>> (len = npackf)
Packf *packf;
int npackf;
```

17.2.2 Pack Index

```
<Pack idx fields 152c>≡ (152a)
// content of .idx file (size = nidx), =~ array<Hash, file-offset-in-pack buf>
// ref_own<string>
char *idx;
vlong nidx;
```

17.2.3 Meta and Metavec

```
<struct Meta 152d>≡ (277)
struct Meta {
    Object *obj;
    vlong path;
    vlong mtime;

    /* The best delta we picked */
    Meta *head;
    Meta *prev;

    <Meta delta fields 153d>
    <Meta dtab field 154a>

    /* Only used for writing offset deltas */
    vlong off;
};
```

```
<function writeordercmp 152e>≡ (277)
static int
writeordercmp(void *pa, void *pb)
{
    Meta *a, *b, *ahd, *bhd;

    a = *(Meta**)pa;
    b = *(Meta**)pb;
    ahd = (a->head == nil) ? a : a->head;
    bhd = (b->head == nil) ? b : b->head;
    if(ahd->mtime != bhd->mtime)
```

```

    return bhd->mtime - ahd->mtime;
if(ahd != bhd)
    return (uintptr)bhd - (uintptr)ahd;
if(a->nchain != b->nchain)
    return a->nchain - b->nchain;
return a->mtime - b->mtime;
}

```

Uses GCommit 33a, addmeta() 173b, loadtree() 173b, osfind() 38c, readobject() 157b, and unref() 34a.

```

⟨function freemeta 153a⟩≡ (277)
static void
freemeta(Meta *m)
{
    free(m->delta);
    free(m);
}

```

```

⟨function deltaordercmp 153b⟩≡ (277)
static int
deltaordercmp(void *pa, void *pb)
{
    Meta *a, *b;
    vlong cmp;

    a = *(Meta**)pa;
    b = *(Meta**)pb;
    if(a->obj->type != b->obj->type)
        return a->obj->type - b->obj->type;
    cmp = (b->path - a->path);
    if(cmp != 0)
        return (cmp < 0) ? -1 : 1;
    cmp = a->mtime - b->mtime;
    if(cmp != 0)
        return (cmp < 0) ? -1 : 1;
    return memcmp(a->obj->hash.h, b->obj->hash.h, sizeof(a->obj->hash.h));
}

```

Uses GBlob 33a, GTree 33a, addmeta() 173b, clearedobject() 43b, loadtree() 173b, murmurhash2() 224, and unref() 34a.

```

⟨struct Metavec 153c⟩≡ (277)
struct Metavec {
    // growing_array<ref_own<Meta>> (used = nmeta, allocated = metasz)
    Meta **meta;
    int nmeta;
    int metasz;
};

```

17.2.4 Delta

```

⟨Meta delta fields 153d⟩≡ (152d)
// ref_own<Delta>
Delta *delta;
int ndelta;
int nchain;

```

```

⟨struct Delta 153e⟩≡ (270)
struct Delta {
    int cpy;
    int off;
    int len;
};

```

17.2.5 Dtab and Dblock

`<Meta dtab field 154a>`≡ (152d)
/* Only used for delta window */
Dtab dtab;

`<struct Dtab 154b>`≡ (270)
struct Dtab {
 Object *o;
 uchar *base;
 int nbase;

 Dblock *b;
 int nb;
 int sz;
};

`<struct Dblock 154c>`≡ (270)
struct Dblock {
 uchar *buf;
 int len;
 int off;
 u64int hash;
};

`<function dtclear 154d>`≡ (266b)
void
dtclear(Dtab *dt)
{
 unref(dt->o);
 free(dt->b);
}

Uses `unref()` 34a.

`<function dtinit 154e>`≡ (266b)
void
dtinit(Dtab *dt, Object *obj)
{
 uchar *s, *e;
 u64int h;
 vlong n, o;

 o = 0;
 s = (uchar*)obj->data;
 e = s + obj->size;
 dt->o = ref(obj);
 dt->nb = 0;
 dt->sz = 128;
 dt->b = eamalloc(dt->sz, sizeof(Dblock));
 dt->base = (uchar*)obj->data;
 dt->nbase = obj->size;

 while(s != e){
 n = nextblk(s, e);
 h = hash(s, n);
 addblk(dt, s, n, o, h);
 s += n;
 o += n;
 }
}

Uses `addblk()` 155b, `eamalloc()`, `hash()` 168b, `nextblk()` 155a, and `ref()` 36c.

```

⟨function nextblk 155a⟩≡ (266b)
static int
nextblk(uchar *s, uchar *e)
{
    u32int gh;
    uchar *p;

    if((e - s) < Minchunk)
        return e - s;
    p = s + Minchunk;
    if((e - s) > Maxchunk)
        e = s + Maxchunk;
    gh = 0;
    while(p != e){
        gh = (gh<<1) + geartab[*p++];
        if((gh & Splitmask) == 0)
            break;
    }
    return p - s;
}

```

Uses Maxchunk-42 266b, Minchunk-41 266b, Splitmask-43 266b, and geartab-44 266b.

```

⟨function addblk 155b⟩≡ (266b)
static void
addblk(Dtab *dt, void *buf, int len, int off, u64int h)
{
    int i, sz, probe;
    Dblock *db;

    probe = h % dt->sz;
    while(dt->b[probe].buf != nil){
        if(len == dt->b[probe].len && memcmp(buf, dt->b[probe].buf, len) == 0)
            return;
        probe = (probe + 1) % dt->sz;
    }
    assert(dt->b[probe].buf == nil);
    dt->b[probe].buf = buf;
    dt->b[probe].len = len;
    dt->b[probe].off = off;
    dt->b[probe].hash = h;
    dt->nb++;
    if(dt->sz < 2*dt->nb){
        sz = dt->sz;
        db = dt->b;
        dt->sz *= 2;
        dt->nb = 0;
        dt->b = eamalloc(dt->sz, sizeof(Dblock));
        for(i = 0; i < sz; i++)
            if(db[i].buf != nil)
                addblk(dt, db[i].buf, db[i].len, db[i].off, db[i].hash);
        free(db);
    }
}

```

Uses addblk() 155b and eamalloc().

17.3 Searching the pack index

```

⟨function searchindex 155c⟩≡ (277)

```

```

vlong
searchindex(char *idx, int nidx, Hash h, int npfx, Hash *hret)
{
    uint lo, hi, hidx;
    vlong o, oo, nent;
    int i, r;
    void *s;

    o = 8;
    if(nidx < 8 + 256*4)
        return -1;
    if(npfx < 8)
        return -1;
    /*
     * Read the fanout table. The fanout table
     * contains 256 entries, corresponding to
     * the first byte of the hash. Each entry
     * is a 4 byte big endian integer, containing
     * the total number of entries with a leading
     * byte <= the table index, allowing us to
     * rapidly do a binary search on them.
     */
    if (h.h[0] == 0){
        lo = 0;
        hi = GETBE32(idx + o);
    } else {
        o += h.h[0]*4 - 4;
        lo = GETBE32(idx + o);
        hi = GETBE32(idx + o + 4);
    }
    if(hi == lo)
        goto notfound;
    nent = GETBE32(idx + 8 + 255*4);

    /*
     * Now that we know the range of hashes that the
     * entry may exist in, search them
     */
    r = -1;
    s = nil;
    hidx = -1;
    o = 8 + 256*4;
    while(lo < hi){
        hidx = (hi + lo)/2;
        s = idx + o + hidx*sizeof(h.h);
        r = hashcmp(h.h, s, npfx);
        if(r < 0)
            hi = hidx;
        else if(r > 0)
            lo = hidx + 1;
        else
            break;
    }
    if(r != 0)
        goto notfound;
    if(hret != nil)
        memcpy(hret, s, sizeof(Hash));

    /*
     * We found the entry. If it's 32 bits, then we

```

```

    * can just return the oset, otherwise the 32
    * bit entry contains the oset to the 64 bit
    * entry.
    */
oo = 8;          /* Header */
oo += 256*4;     /* Fanout table */
oo += Hashsz*nent; /* Hashes */
oo += 4*nent;    /* Checksums */
oo += 4*hidx;    /* Offset offset */
if(oo < 0 || oo + 4 > nidx)
    goto err;
i = GETBE32(idx + oo);
o = i & 0xffffffffULL;
/*
 * Large offsets (i.e. 64-bit) are encoded as an index
 * into the next table with the MSB bit set.
 */
if(o & (1ull << 31)){
    o &= 0x7fffffffULL;
    oo = 8;          /* Header */
    oo += 256*4;     /* Fanout table */
    oo += Hashsz*nent; /* Hashes */
    oo += 4*nent;    /* Checksums */
    oo += 4*nent;    /* 32-bit Offsets */
    oo += 8*o;       /* 64-bit Offset offset */
    if(oo < 0 || oo + 8 >= nidx)
        goto err;
    o = GETBE64(idx + oo);
}
return o;

err:
    werrstr("out of bounds read");
    return -1;
notfound:
    werrstr("not present");
    return -1;
}

```

Uses GETBE64 270 and Hashsz 161c.

```

<macro GETBE64 157a>≡ (270)
#define GETBE64(b)\
    (((b)[0] & 0xFFull) << 56) | \
    (((b)[1] & 0xFFull) << 48) | \
    (((b)[2] & 0xFFull) << 40) | \
    (((b)[3] & 0xFFull) << 32) | \
    (((b)[4] & 0xFFull) << 24) | \
    (((b)[5] & 0xFFull) << 16) | \
    (((b)[6] & 0xFFull) << 8) | \
    (((b)[7] & 0xFFull) << 0)

```

17.4 readidxobject() part 2

```

<readidxobject() look for object in packs 157b>≡ (43b)
for(i = 0; i < npackf; i++){
    o = searchindex(packf[i].idx, packf[i].nidx, h, SHA1dlen*8, nil);
    if(o != ERROR_NEG1){
        f = openpack(&packf[i]);
    }
}

```

```

    if(f == nil)
        goto error;
    r = Bseek(f, o, 0);
    if(r != ERROR_NEG1)
        r = readpacked(f, obj, flag);
    closepack(&packf[i]);
    if(r == ERROR_NEG1)
        goto error;
    parseobject(obj);
    cache(obj);
    return obj;
}
}

```

Uses `parseobject()` 48c, `readidxobject()` 47c, and `ref()` 36c.

```

⟨readidxobject() when object not found call refreshpacks() 158a⟩≡ (43b)
    refreshpacks();

```

Uses `Cexist`.

```

⟨expandprefix() call refreshpacks() 158b⟩≡ (52a)
    refreshpacks();

```

```

⟨expandprefix() iterate on packf 158c⟩≡ (52a)
    for(i = 0; i < npackf; i++)
        if(searchindex(packf[i].idx, packf[i].nid, h, npfx, rh) != ERROR_NEG1)
            return OK_0;

```

17.5 Reading packs

17.5.1 Refreshing packs and loading indexes

```

⟨function refreshpacks 158d⟩≡ (277)
    /// readidxobject | expandprefix -> <> -> loadpack
    static void
    refreshpacks(void)
    {
        Packf *pf, *new;
        int i, n, l;
        int nnew;
        Dir *d;

        n = slurpdir(".git/objects/pack", &d);
        ⟨refreshpacks() sanity check n 159b⟩
        nnew = 0;
        new = eamalloc(n, sizeof(Packf));

        for(i = 0; i < n; i++){
            l = strlen(d[i].name);
            if(l > 4 && strcmp(d[i].name + l - 4, ".idx") != 0)
                continue;
            d[i].name[l - 4] = '\0';
            if(loadpack(&new[nnew], d[i].name) != -1)
                nnew++;
        }
        ⟨refreshpacks() free previous packf 159a⟩
        packf = new;
        npackf = nnew;
        free(d);
    }

```

```
}
```

Uses Npackcache, openpacks 152c, and packf 152c.

```
<refreshpacks() free previous packf 159a>≡ (158d)
// free the previously loaded one if any
for(i = 0; i < npackf; i++){
    pf = &packf[i];
    free(pf->idx);
    if(pf->pack != nil)
        Bterm(pf->pack);
}
free(packf);
```

Uses openpacks 152c.

```
<refreshpacks() sanity check n 159b>≡ (158d)
if(n == ERROR_NEG1)
    return;
```

Uses packf 152c.

```
<function loadpack 159c>≡ (277)
/// refreshpacks -> <>
static errorneg1
loadpack(Packf *pf, char *name)
{
    char buf[128];
    int i;
    fdt ifd;
    Dir *d;

    memset(pf, 0, sizeof(Packf));
    snprintf(buf, sizeof(buf), ".git/objects/pack/%s.idx", name);
    snprintf(pf->path, sizeof(pf->path), ".git/objects/pack/%s.pack", name);
    <loadpack() steal loaded info from previous opened pack if same 160b>
    // else
    ifd = open(buf, OREAD);
    <loadpack() sanity check ifd 159d>
    d = dirfstat(ifd);
    <loadpack() sanity check d 160a>
    pf->nidx = d->length;
    pf->idx = emalloc(pf->nidx);
    if(readn(ifd, pf->idx, pf->nidx) != pf->nidx){
        close(ifd);
        free(pf->idx);
        free(d);
        return ERROR_NEG1;
    }
    close(ifd);
    free(d);
    return OK_0;
}
```

```
<loadpack() sanity check ifd 159d>≡ (159c)
if(ifd == ERROR_NEG1)
    return ERROR_NEG1;
```

Uses packf 152c.

```

⟨loadpack() sanity check d 160a⟩≡ (159c)
    if(d == nil){
        close(ifd);
        return ERROR_NEG1;
    }

```

Uses `npackf 152c` and `packf 152c`.

```

⟨loadpack() steal loaded info from previous opened pack if same 160b⟩≡ (159c)
/*
 * if we already have the pack open, just
 * steal the loaded info
 */
for(i = 0; i < npackf; i++){
    if(strcmp(pf->path, packf[i].path) == 0){
        pf->idx = packf[i].idx;
        pf->nidx = packf[i].nidx;
        pf->pack = packf[i].pack;
        packf[i].idx = nil;
        packf[i].pack = nil;
        return OK_0;
    }
}

```

Uses `eamalloc()` and `slurpdir() 260`.

17.5.2 Opening a pack

```

⟨Pack extra fields 160c⟩≡ (152a) 161a▷
// openpack()/closepack() reference counting
int refs;

```

```

⟨function openpack 160d⟩≡ (277)
static Biobuf*
openpack(Packf *pf)
{
    ⟨openpack() locals 160f⟩

    if(pf->pack != nil){
        pf->refs++;
        return pf->pack;
    }
    // else
    ⟨openpack() if too many opened packs 161d⟩
    pf->refs++;
    pf->pack = Bopen(pf->path, OREAD);
    return pf->pack;
}

```

```

⟨function closepack 160e⟩≡ (277)
static void
closepack(Packf *pf)
{
    pf->refs--;
}

```

```

⟨openpack() locals 160f⟩≡ (160d)
vlong t;
int i, best;

```

```

⟨Pack extra fields 161a⟩+≡ (152a) <160c
    // for LRU eviction of opened packs
    vlong opentm;

⟨global openpacks 161b⟩≡ (277)
    int openpacks;

⟨constant Npackcache 161c⟩≡ (270)
    Npackcache = 32,

⟨openpack() if too many opened packs 161d⟩≡ (160d)
    /*
    * If we've got more packs open
    * than we want cached, try to
    * free up the oldest ones.
    *
    * If we can't find a slot, this
    * isn't fatal; we can just use
    * another fd.
    */
    while(openpacks >= Npackcache){
        t = (1ull<<62)-1;
        best = -1;
        for(i = 0; i < npackf; i++){
            if(&packf[i] != pf
                && packf[i].pack != nil
                && packf[i].opentm < t
                && packf[i].refs == 0){
                t = packf[i].opentm;
                best = i;
            }
        }
        if(best == -1){
            fprintf(STDERR, "no available pack slots\n");
            break;
        }
        Bterm(packf[best].pack);
        packf[best].pack = nil;
        openpacks--;
    }
    openpacks++;
    pf->opentm = nsec();

```

17.5.3 Reading a packed object

```

⟨function readpacked 161e⟩≡ (277)
    /// readidxobject -> <>
    static errorneg1
    readpacked(Biobuf *f, Object *o, int flag)
    {
        int c, s, n, t;
        vlong l, p;
        Buf b;

        p = Boffset(f);
        c = Bgetc(f);
        ⟨readpacked() sanity check c 162a⟩
        l = c & 0xf;
        s = 4;

```

```

t = (c >> 4) & 0x7;
⟨readpacked() sanity check t 162b⟩
while(c & 0x80){
    if((c = Bgetc(f)) == -1)
        return ERROR_NEG1;
    l |= (vlong)(c & 0x7f) << s;
    s += 7;
}
⟨readpacked() sanity check l 162c⟩
switch(t){
⟨readpacked() switch object type t cases 162d⟩
default:
    werrstr("invalid object at %lld", Boffset(f));
    return ERROR_NEG1;
}
o->flag |= Cloaded|flag;
return OK_0;
}

```

Uses Cloaded, GCommit 33a, GTag 177b, and decompress() 276b.

```

⟨readpacked() sanity check c 162a⟩≡ (161e)
if(c == -1)
    return ERROR_NEG1;

```

Uses GBlob 33a.

```

⟨readpacked() sanity check t 162b⟩≡ (161e)
if(!t){
    werrstr("unknown type for byte %x at %lld", c, p);
    return ERROR_NEG1;
}

```

```

⟨readpacked() sanity check l 162c⟩≡ (161e)
if(l >= (1ULL << 32)){
    werrstr("object too big");
    return ERROR_NEG1;
}

```

Uses GNone 270.

```

⟨readpacked() switch object type t cases 162d⟩≡ (161e) 177c▷
case GCommit:
case GTree:
case GTag:
case GBlob:
    b.sz = 64 + 1;
    b.data = emalloc(b.sz);
    n = snprintf(b.data, 64, "%T %lld", t, l) + 1;
    b.len = n;
    if(bdecompress(&b, f, nil) == ERROR_NEG1){
        free(b.data);
        return ERROR_NEG1;
    }
    o->len = Boffset(f) - o->off;
    o->type = t;
    o->all = b.data;
    o->data = b.data + n;
    o->size = b.len - n;
    break;

```

Uses GNone 270.

⟨Object *indexing fields* 163a⟩≡

(32b) 177a▷

```
    vlong   off;
    vlong   len;
```

17.6 Writing packs

17.6.1 writepack()

⟨function *writepack* 163b⟩≡

(277)

```
    /// main(repack.c) | (main(send.c) -> sendpack) | servpack -> <> -> genpack
    errorneg1
    writepack(fdt fd, Hash *theirs, int ntheirs, Hash *ours, int nours, Hash *h)
    {
        int i;
        Meta **meta;
        int nmeta;
        errorneg1 r;

        if((nmeta = readmeta(theirs, ntheirs, ours, nours, &meta)) == -1)
            return ERROR_NEG1;
        pickdeltas(meta, nmeta);
        r = genpack(fd, meta, nmeta, h, false);
        for(i = 0; i < nmeta; i++)
            freemeta(meta[i]);
        free(meta);
        return r;
    }
```

17.6.2 readmeta()

⟨function *readmeta* 163c⟩≡

(277)

```
    static int
    readmeta(Hash *theirs, int ntheirs, Hash *ours, int nours, Meta ***m)
    {
        Object **obj;
        Objset has;
        int i, nobj;
        Metavec v;

        *m = nil;
        osinit(&has);
        v.nmeta = 0;
        v.metasz = 64;
        v.meta = eamalloc(v.metasz, sizeof(Meta*));
        if(findtwixt(theirs, ntheirs, ours, nours, &obj, &nobj) == -1)
            sysfatal("load twixt: %r");

        if(nobj == 0)
            return 0;
        for(i = 0; i < nours; i++)
            if(!hasheq(&ours[i], &Zhash))
                if(loadcommit(nil, &has, ours[i]) == -1)
                    goto out;
        for(i = 0; i < nobj; i++)
            if(loadcommit(&v, &has, obj[i]->hash) == -1)
                goto out;
        osclear(&has);
```

```

    *m = v.meta;
    return v.nmeta;
out:
    osclear(&has);
    free(v.meta);
    return -1;
}

```

Uses dtclear() 154d, hwrite() 52a, interactive, and max-39 256a.

findtwixt()

```

⟨function findtwixt 164a⟩≡ (281)
    int
    findtwixt(Hash *head, int nhead, Hash *tail, int ntail, Object ***res, int *nres)
    {
        return paint(head, nhead, tail, ntail, res, nres, Twixt);
    }

```

```

⟨PaintMode other cases 164b⟩≡ (95d)
    Twixt,

```

```

⟨paint() switch mode cases 164c⟩+≡ (96) <95c
    case Twixt:
        dprint(1, "found twixt\n");
        *res = eamalloc(keep.nobj, sizeof(Object*));
        *nres = 0;
        for(i = 0; i < keep.sz; i++){
            o = keep.obj[i];
            if(o != nil && !oshas(&drop, o->hash) && !oshas(&skip, o->hash)){
                (*res)[*nres] = o;
                (*nres)++;
            }
        }
        break;

```

loadcommit()

```

⟨function loadcommit 164d⟩≡ (277)
    static int
    loadcommit(Metavec *v, Objset *has, Hash h)
    {
        Object *c;
        int r;

        if(osfind(has, h))
            return 0;
        if((c = readobject(h)) == nil)
            return -1;
        if(c->type != GCommit){
            fprintf(2, "load: %H: not commit\n", c->hash);
            unref(c);
            return 0;
        }
        addmeta(v, has, c, 0, c->commit->ctime);
        r = loadtree(v, has, c->commit->tree, "", c->commit->ctime);
        unref(c);
        return r;
    }

```

Uses deltasz() 165b and deltify() 167b.

addmeta()

```
<function addmeta 165a>≡ (277)
static void
addmeta(Metavec *v, Objset *has, Object *o, vlong pathid, vlong mtime)
{
    Meta *m;

    if(oshas(has, o->hash))
        return;
    osadd(has, o);
    if(v == nil)
        return;
    m = emalloc(sizeof(Meta));
    m->obj = o;
    m->path = pathid;
    m->mtime = mtime;

    if(v->nmeta == v->metasz){
        v->metasz = 2*v->metasz;
        v->meta = earealloc(v->meta, v->metasz, sizeof(Meta*));
    }
    v->meta[v->nmeta++] = m;
}
```

loadtree()

```
<function loadtree 165b>≡ (277)
static int
loadtree(Metavec *v, Objset *has, Hash tree, char *dpath, vlong mtime)
{
    Object *t, *o;
    Dirent *e;
    vlong dh, eh;
    int i, k, r;
    char *p;

    if(oshas(has, tree))
        return 0;
    if((t = readobject(tree)) == nil)
        return -1;
    if(t->type != GTree){
        fprintf(2, "load: %H: not tree\n", t->hash);
        unref(t);
        return 0;
    }
    dh = murmurhash2(dpath, strlen(dpath));
    addmeta(v, has, t, dh, mtime);
    for(i = 0; i < t->tree->nent; i++){
        e = &t->tree->ent[i];
        if(oshas(has, e->h))
            continue;
        if(e->ismod)
            continue;
        k = (e->mode & DMDIR) ? GTree : GBlob;
        o = clearedobject(e->h, k);
        if(k == GTree){
            p = smprint("%s/%s", dpath, e->name);
            r = loadtree(v, has, e->h, p, mtime);
            free(p);
        }
    }
}
```

```

        if(r == -1)
            return -1;
    }else{
        eh = murmurhash2(e->name, strlen(e->name));
        addmeta(v, has, o, dh^eh, mtime);
    }
}
unref(t);
return 0;
}

```

Uses GCommit 33a, GTag 177b, deltaordercmp() 173b, dprint 270, dtclear() 154d, dtinit() 154e, interactive, readobject() 157b, and showprogress() 39d.

17.6.3 pickdeltas()

(function pickdeltas 166)≡ (277)

```

static void
pickdeltas(Meta **meta, int nmeta)
{
    Meta *m, *p;
    Object *o;
    Delta *d;
    int i, j, nd, sz, pct, best;

    pct = 0;
    dprint(1, "picking deltas\n");
    if(interactive)
        fprintf(2, "deltifying %d objects: 0%%", nmeta);
    qsort(meta, nmeta, sizeof(Meta*), deltaordercmp);

    for(i = 0; i < nmeta; i++){
        m = meta[i];
        pct = showprogress((i*100) / nmeta, pct);

        m->delta = nil;
        m->ndelta = 0;
        if(m->obj->type == GCommit || m->obj->type == GTag)
            continue;
        if((o = readobject(m->obj->hash)) == nil)
            sysfatal("readobject %H: %r", m->obj->hash);
        dtinit(&m->dtab, o);
        if(i >= 11)
            dtclear(&meta[i-11]->dtab);
        best = o->size;
        for(j = max(0, i - 10); j < i; j++){
            p = meta[j];
            /* long chains make unpacking slow */
            if(p->nchain >= 128 || p->obj->type != o->type)
                continue;
            d = deltify(o, &p->dtab, &nd);
            sz = deltasz(d, nd);
            if(sz + 32 < best){
                /*
                 * if we already picked a best delta,
                 * replace it.
                 */
                free(m->delta);
                best = sz;
                m->delta = d;
            }
        }
    }
}

```

```

        m->ndelta = nd;
        m->nchain = p->nchain + 1;
        m->prev = p;
        m->head = p->head;
        if(m->head == nil)
            m->head = p;
    }else
        free(d);
    }
    unref(o);
}
for(i = max(0, nmeta - 10); i < nmeta; i++)
    dtclear(&meta[i]->dtab);
if(interactive)
    fprintf(2, "\\b\\b\\b\\b100%%\n");
}

```

<function deltasz 167a>≡ (277)

```

static int
deltasz(Delta *d, int nd)
{
    int i, sz;
    sz = 32;
    for(i = 0; i < nd; i++)
        sz += d[i].cpy ? 7 : d[i].len + 1;
    return sz;
}

```

deltify()

<function deltify 167b>≡ (266b)

```

Delta*
deltify(Object *obj, Dtab *dt, int *pnd)
{
    Delta *d;
    Dblock *b;
    uchar *s, *e;
    vlong n, o;

    o = 0;
    d = nil;
    s = (uchar*)obj->data;
    e = s + obj->size;
    *pnd = 0;
    while(s != e){
        n = nextblk(s, e);
        b = lookup(dt, s, n);
        n = stretch(dt, b, s, e, n);
        if(b != nil)
            emitdelta(&d, pnd, 1, b->off, n);
        else
            emitdelta(&d, pnd, 0, o, n);
        s += n;
        o += n;
    }
    return d;
}

```

Uses emitdelta() 169a, nextblk() 155a, and stretch() 168c.

lookup()

```
<function lookup 168a>≡ (266b)
static Dblock*
lookup(Dtab *dt, uchar *p, int n)
{
    int probe;
    u64int h;

    h = hash(p, n);
    for(probe = h % dt->sz; dt->b[probe].buf != nil; probe = (probe + 1) % dt->sz){
        if(dt->b[probe].hash != h)
            continue;
        if(n != dt->b[probe].len)
            continue;
        if(memcmp(p, dt->b[probe].buf, n) != 0)
            continue;
        return &dt->b[probe];
    }
    return nil;
}
```

Uses hash() 168b.

```
<function hash 168b>≡ (266b)
static u64int
hash(void *p, int n)
{
    return murmurhash2(p, n);
}
```

Uses murmurhash2() 224.

stretch()

```
<function stretch 168c>≡ (266b)
static int
stretch(Dtab *dt, Dblock *b, uchar *s, uchar *e, int n)
{
    uchar *p0, *p, *q, *eb;

    if(b == nil)
        return n;
    p = s + n;
    q = dt->base + b->off + n;
    p0 = p;
    if(dt->nbase < (1<<24)-1)
        eb = dt->base + dt->nbase;
    else
        eb = dt->base + (1<<24)-1;
    while(true){
        if(p == e || q == eb)
            break;
        if(*p != *q)
            break;
        p++;
        q++;
    }
    return n + (p - p0);
}
```

emitdelta()

```
<function emitdelta 169a>≡ (266b)
static int
emitdelta(Delta **pd, int *nd, int cpy, int off, int len)
{
    Delta *d;

    *nd += 1;
    *pd = earealloc(*pd, *nd, sizeof(Delta));
    d = &(*pd)[*nd - 1];
    d->cpy = cpy;
    d->off = off;
    d->len = len;
    return len;
}
```

Uses earealloc() 257a.

17.6.4 genpack()

```
<function genpack 169b>≡ (277)
/// writepack(odelta = false) -> <>
static int
genpack(fdt fd, Meta **meta, int nmeta, Hash *h, bool odelta)
{
    int i, nh, nd, res, pct, ret;
    DigestState *st;
    Biobuf *bfd;
    Meta *m;
    Object *o, *po, *b;
    char *p, buf[32];

    st = nil;
    ret = -1;
    pct = 0;
    dprint(1, "generating pack\n");
    if((fd = dup(fd, -1)) == -1)
        return -1;
    if((bfd = Bfdopen(fd, OWRITE)) == nil)
        return -1;
    if(hwrite(bfd, "PACK", 4, &st) == -1)
        return -1;
    PUTBE32(buf, 2);
    if(hwrite(bfd, buf, 4, &st) == -1)
        return -1;
    PUTBE32(buf, nmeta);
    if(hwrite(bfd, buf, 4, &st) == -1)
        return -1;

    qsort(meta, nmeta, sizeof(Meta*), writeordercmp);

    if(interactive)
        fprintf(2, "writing %d objects: 0%%", nmeta);

    for(i = 0; i < nmeta; i++){
        pct = showprogress((i*100)/nmeta, pct);
        m = meta[i];
        m->off = Boffset(bfd);
        if(m->off == -1)

```

```

        goto error;
if((o = readobject(m->obj->hash)) == nil)
    return -1;
if(m->delta == nil){
    nh = packhdr(buf, o->type, o->size);
    if(hwrite(bfd, buf, nh, &st) == -1)
        goto error;
    if(hcompress(bfd, o->data, o->size, &st) == -1)
        goto error;
}else{
    if((b = readobject(m->prev->obj->hash)) == nil)
        goto error;
    nd = encodedelta(m, o, b, &p);
    unref(b);
    if(odelta && m->prev->off != 0){
        nh = 0;
        nh += packhdr(buf, G0delta, nd);
        nh += packoff(buf+nh, m->off - m->prev->off);
        if(hwrite(bfd, buf, nh, &st) == -1)
            goto error;
    }else{
        nh = packhdr(buf, GRdelta, nd);
        po = m->prev->obj;
        if(hwrite(bfd, buf, nh, &st) == -1)
            goto error;
        if(hwrite(bfd, po->hash.h, sizeof(po->hash.h), &st) == -1)
            goto error;
    }
    res = hcompress(bfd, p, nd, &st);
    free(p);
    if(res == -1)
        goto error;
}
unref(o);
}
if(interactive)
    fprintf(2, "\b\b\b\b100%\n");
sha1(nil, 0, h->h, st);
if(Bwrite(bfd, h->h, sizeof(h->h)) == -1)
    goto error;
ret = 0;
error:
    if(Bterm(bfd) == -1)
        return -1;
    return ret;
}

```

<macro PUTBE32 170a>≡ (270)

```

#define PUTBE32(b, n)\
do{ \
    (b)[0] = (n) >> 24; \
    (b)[1] = (n) >> 16; \
    (b)[2] = (n) >> 8; \
    (b)[3] = (n) >> 0; \
} while(0)

```

packhdr(), packoff()

<function packhdr 170b>≡ (277)

```

static int
packhdr(char *hdr, int ty, int len)
{
    int i;

    hdr[0] = ty << 4;
    hdr[0] |= len & 0xf;
    len >>= 4;
    for(i = 1; len != 0; i++){
        hdr[i-1] |= 0x80;
        hdr[i] = len & 0x7f;
        len >>= 7;
    }
    return i;
}

```

Uses interactive and unref() 34a.

<function packoff 171a>≡ (277)

```

static int
packoff(char *hdr, vlong off)
{
    int i, j;
    char rbuf[8];

    rbuf[0] = off & 0x7f;
    for(i = 1; (off >>= 7) != 0; i++)
        rbuf[i] = (--off & 0x7f)|0x80;

    j = 0;
    while(i > 0)
        hdr[j++] = rbuf[--i];
    return j;
}

```

Uses freemeta() 173b, genpack() 172c, and pickdeltas() 165b.

hcompress()

<struct Compout 171b>≡ (277)

```

struct Compout {
    Biobuf *bfd;
    DigestState *st;
};

```

<function hcompress 171c>≡ (277)

```

static int
hcompress(Biobuf *bfd, void *buf, int sz, DigestState **st)
{
    int r;
    Buf b ={
        .off=0,
        .data=buf,
        .sz=sz,
    };
    Compout o = {
        .bfd = bfd,
        .st = *st,
    };
    r = deflatezlib(&o, compwrite, &b, compread, 6, 0);
}

```

```

    *st = o.st;
    return r;
}

```

<function compread 172a>≡ (277)

```

static int
compread(void *p, void *dst, int n)
{
    Buf *b;

    b = p;
    if(n > b->sz - b->off)
        n = b->sz - b->off;
    memcpy(dst, b->data + b->off, n);
    b->off += n;
    return n;
}

```

<function compwrite 172b>≡ (277)

```

static int
compwrite(void *p, void *buf, int n)
{
    return hwrite(((Compout *)p)->bfd, buf, n, &(((Compout*)p)->st);
}

```

encodedelta()

<function encodedelta 172c>≡ (277)

```

static int
encodedelta(Meta *m, Object *o, Object *b, void **pp)
{
    char *p, *bp, buf[16];
    int len, sz, n, i, j;
    Delta *d;

    sz = 128;
    len = 0;
    p = emalloc(sz);

    /* base object size */
    buf[0] = b->size & 0x7f;
    n = b->size >> 7;
    for(i = 1; n > 0; i++){
        buf[i - 1] |= 0x80;
        buf[i] = n & 0x7f;
        n >>= 7;
    }
    append(&p, &len, &sz, buf, i);

    /* target object size */
    buf[0] = o->size & 0x7f;
    n = o->size >> 7;
    for(i = 1; n > 0; i++){
        buf[i - 1] |= 0x80;
        buf[i] = n & 0x7f;
        n >>= 7;
    }
    append(&p, &len, &sz, buf, i);
    for(j = 0; j < m->ndelta; j++){

```

```

d = &m->delta[j];
if(d->cpy){
    n = d->off;
    bp = buf + 1;
    buf[0] = 0x81;
    buf[1] = 0x00;
    for(i = 0; i < sizeof(buf); i++) {
        buf[0] |= 1<<i;
        *bp++ = n & 0xff;
        n >>= 8;
        if(n == 0)
            break;
    }

    n = d->len;
    if(n != 0x10000) {
        buf[0] |= 0x1<<4;
        for(i = 0; i < sizeof(buf)-4 && n > 0; i++){
            buf[0] |= 1<<(i + 4);
            *bp++ = n & 0xff;
            n >>= 8;
        }
    }
    append(&p, &len, &sz, buf, bp - buf);
}else{
    n = 0;
    while(n != d->len){
        buf[0] = (d->len - n < 127) ? d->len - n : 127;
        append(&p, &len, &sz, buf, 1);
        append(&p, &len, &sz, o->data + d->off + n, buf[0]);
        n += buf[0];
    }
}
}
*pp = p;
return len;
}

```

Uses `G0delta` 33a, `GRdelta` 33a, `PUTBE32` 32a, `dprint` 270, `encodedelta()` 166, `hcompress()`, `hwrite()` 52a, `interactive`, `packhdr()` 171c, `packoff()` 173a, `readobject()` 157b, `showprogress()` 39d, `unref()` 34a, and `writeordercmp()` 173b.

```

⟨function append 173a⟩≡ (277)
static void
append(char **p, int *len, int *sz, void *seg, int nseg)
{
    if(*len + nseg >= *sz){
        while(*len + nseg >= *sz)
            *sz += *sz/2;
        *p = erealloc(*p, *sz);
    }
    memcpy(*p + *len, seg, nseg);
    *len += nseg;
}

```

17.7 Indexing packs

17.7.1 `indexpack()`

```

⟨function indexpack 173b⟩≡ (277)

```

```

/// main(repack.c) | fetchpack -> <>
errorneg1
indexpack(char *pack, char *idx, Hash ph)
{
    char hdr[4*3], buf[8];
    int nobj, npct, nvalid, nbig;
    int i, n, pct;
    Object *o, **obj;
    DigestState *st;
    char *valid;
    Biobuf *f;
    Hash h;
    int c;

    f = Bopen(pack, OREAD);
    if(f == nil)
        return -1;
    if(Bread(f, hdr, sizeof(hdr)) != sizeof(hdr)){
        werrstr("short read on header");
        return -1;
    }
    if(memcmp(hdr, "PACK\0\0\0\2", 8) != 0){
        werrstr("invalid header");
        return -1;
    }

    pct = 0;
    npct = 0;
    nvalid = 0;
    nobj = GETBE32(hdr + 8);
    obj = eamalloc(nobj, sizeof(Object*));
    valid = eamalloc(nobj, sizeof(char));
    if(interactive)
        fprintf(STDERR, "indexing %d objects: 0%%", nobj);
    while(nvalid != nobj){
        n = 0;
        for(i = 0; i < nobj; i++){
            if(valid[i]){
                n++;
                continue;
            }
        }
        pct = showprogress((npct*100)/nobj, pct);
        if(obj[i] == nil){
            o = emalloc(sizeof(Object));
            o->off = Boffset(f);
            obj[i] = o;
        }
        o = obj[i];
        /*
         * We can seek around when packing delta chains.
         * Be extra careful while we don't know where all
         * the objects start.
         */
        Bseek(f, o->off, 0);
        if(readpacked(f, o, Cidx) == -1)
            continue;
        sha1((uchar*)o->all, o->size + strlen(o->all) + 1, o->hash.h, nil);
        valid[i] = 1;
        cache(o);
        npct++;
    }
}

```

```

        n++;
        if(objectcrc(f, o) == -1)
            return -1;
    }
    if(n == nvalid)
        sysfatal("fix point reached too early: %d/%d: %r", nvalid, nobj);
    nvalid = n;
}
if(interactive)
    fprintf(2, "\b\b\b\b100%\n");
Bterm(f);

st = nil;
qsort(obj, nobj, sizeof(Object*), objcmp);
if((f = Bopen(idx, OWRITE)) == nil)
    return -1;
if(hwrite(f, "\xff0c\x00\x00\x00", 8, &st) != 8)
    goto error;
/* fanout table */
c = 0;
for(i = 0; i < 256; i++){
    while(c < nobj && (obj[c]->hash.h[0] & 0xff) <= i)
        c++;
    PUTBE32(buf, c);
    if(hwrite(f, buf, 4, &st) == -1)
        goto error;
}
for(i = 0; i < nobj; i++){
    o = obj[i];
    if(hwrite(f, o->hash.h, sizeof(o->hash.h), &st) == -1)
        goto error;
}

for(i = 0; i < nobj; i++){
    PUTBE32(buf, obj[i]->crc);
    if(hwrite(f, buf, 4, &st) == -1)
        goto error;
}

nbig = 0;
for(i = 0; i < nobj; i++){
    if(obj[i]->off < (1ull<<31))
        PUTBE32(buf, obj[i]->off);
    else{
        PUTBE32(buf, (1ull << 31) | nbig);
        nbig++;
    }
    if(hwrite(f, buf, 4, &st) == -1)
        goto error;
}
for(i = 0; i < nobj; i++){
    if(obj[i]->off >= (1ull<<31)){
        PUTBE64(buf, obj[i]->off);
        if(hwrite(f, buf, 8, &st) == -1)
            goto error;
    }
}
if(hwrite(f, ph.h, sizeof(ph.h), &st) == -1)
    goto error;
sha1(nil, 0, h.h, st);

```

```
Bwrite(f, h.h, sizeof(h.h));
```

```
free(obj);  
free(valid);  
Bterm(f);  
return 0;
```

```
error:
```

```
free(obj);  
free(valid);  
Bterm(f);  
return -1;
```

```
}
```

Uses GTree 33a, PUTBE32 32a, PUTBE64 157a, addmeta() 173b, earealloc() 257a, hwrite() 52a, murmurhash2() 224, osadd() 37e, oshas() 38d, readobject() 157b, and unref() 34a.

```
<macro PUTBE64 176a>≡ (270)
```

```
#define PUTBE64(b, n)\  
do{ \  
    (b)[0] = (n) >> 56; \  
    (b)[1] = (n) >> 48; \  
    (b)[2] = (n) >> 40; \  
    (b)[3] = (n) >> 32; \  
    (b)[4] = (n) >> 24; \  
    (b)[5] = (n) >> 16; \  
    (b)[6] = (n) >> 8; \  
    (b)[7] = (n) >> 0; \  
} while(0)
```

```
<function hwrite 176b>≡ (277)
```

```
static int  
hwrite(Biobuf *b, void *buf, int len, DigestState **st)  
{  
    *st = sha1(buf, len, nil, *st);  
    return Bwrite(b, buf, len);  
}
```

Uses interactive and objcmp() 43b.

```
<function showprogress 176c>≡ (284b)
```

```
int  
showprogress(int x, int pct)  
{  
    if(!interactive)  
        return 0;  
    if(x > pct){  
        pct = x;  
        fprintf(STDERR, "\b\b\b\b\b%3d%", pct);  
    }  
    return pct;  
}
```

```
<function objectcrc 176d>≡ (277)
```

```
/// indexpack -> <>  
static u32int  
objectcrc(Biobuf *f, Object *o)  
{  
    char buf[8096];  
    int n, r;  
  
    o->crc = 0;
```

```

Bseek(f, o->off, 0);
for(n = o->len; n > 0; n -= r){
    r = Bread(f, buf, n > sizeof(buf) ? sizeof(buf) : n);
    if(r == -1)
        return -1;
    if(r == 0)
        return 0;
    o->crc = crc32(o->crc, buf, r);
}
return 0;
}

```

Uses PUTBE32 32a and hwrite() 52a.

<Object indexing fields 177a>+≡ (32b) <163a
u32int crc;

17.8 Delta

<Gxxx other cases 177b>≡ (33a) 238a▷
G0delta = 6,
GRdelta = 7,

17.8.1 Odelta

<readpacked() switch object type t cases 177c>+≡ (161e) <162d 179b▷
case G0delta:
if(readodelta(f, o, l, p, flag) == ERROR_NEG1)
return ERROR_NEG1;
break;

Uses Cloded.

<function readodelta 177d>≡ (277)
static int
readodelta(Biobuf *f, Object *o, vlong nd, vlong p, int flag)
{
Object b;
char *d;
vlong r;
int c, n;

d = nil;
if((c = Bgetc(f)) == -1)
return -1;
r = c & 0x7f;
while(c & 0x80 && r < (1ULL<<56)){
if((c = Bgetc(f)) == -1)
return -1;
r = ((r + 1)<<7) | (c & 0x7f);
}

if(r > p || r >= (1ULL<<56)){
werrstr("junk offset -%lld (from %lld)", r, p);
goto error;
}
if((n = decompress(&d, f, nil)) == -1)
goto error;
o->len = Boffset(f) - o->off;

```

if(d == nil || n != nd)
    goto error;
if(Bseek(f, p - r, 0) == -1)
    goto error;
memset(&b, 0, sizeof(Object));
if(readpacked(f, &b, flag) == -1)
    goto error;
if(applydelta(o, &b, d, nd) == -1)
    goto error;
clear(&b);
free(d);
return 0;
error:
    free(d);
    return -1;
}

```

Uses GBlob 33a, GCommit 33a, G0delta 33a, GRdelta 33a, GTag 177b, GTree 33a, bdecompress() 276b, and readodelta() 178.

(function applydelta 178)≡ (277)

```

static int
applydelta(Object *dst, Object *base, char *d, int nd)
{
    char *r, *b, *ed, *er;
    vlong o, l, n, nr, c;

    ed = d + nd;
    b = base->data;
    n = readvint(d, ed, &d);
    if(n == -1 || n >= 1LL << 31 || n != base->size){
        werrstr("mismatched source size");
        return -1;
    }

    nr = readvint(d, ed, &d);
    if(nr == -1 || nr >= 1LL << 31){
        werrstr("invalid pack: %r");
        return -1;
    }
    r = emalloc(nr + 64);
    n = snprintf(r, 64, "%T %lld", base->type, nr) + 1;
    dst->all = r;
    dst->type = base->type;
    dst->data = r + n;
    dst->size = nr;
    er = dst->data + nr;
    r = dst->data;

    while(d != ed){
        c = *d++;
        /* copy from base */
        if(c & 0x80){
            o = 0;
            l = 0;
            /* Offset in base */
            if(d != ed && (c & 0x01)) o |= (*d++ & 0xff) << 0;
            if(d != ed && (c & 0x02)) o |= (*d++ & 0xff) << 8;
            if(d != ed && (c & 0x04)) o |= (*d++ & 0xff) << 16;
            if(d != ed && (c & 0x08)) o |= (*d++ & 0xff) << 24;

            /* Length to copy */

```

```

    if(d != ed && (c & 0x10)) l |= (*d++ & 0xff) << 0;
    if(d != ed && (c & 0x20)) l |= (*d++ & 0xff) << 8;
    if(d != ed && (c & 0x40)) l |= (*d++ & 0xff) << 16;
    if(l == 0) l = 0x10000;

    if(o < 0 || l < 0 || l > er - r || o + l > base->size){
        werrstr("garbled delta: out of bounds copy");
        return -1;
    }
    memmove(r, b + o, l);
    r += l;
    /* inline data */
} else {
    if(c > ed - d || c > er - r){
        werrstr("garbled delta: write past object");
        return -1;
    }
    memmove(r, d, c);
    d += c;
    r += c;
}
}
if(r != er){
    werrstr("truncated delta");
    return -1;
}

return nr;
}

```

Uses Cthin 180a, applydelta() 46e, decompress() 276b, hasheq(), and readidxobject() 47c.

```

⟨function readvint 179a⟩≡ (277)
static vlong
readvint(char *p, char *ep, char **pp)
{
    int s, c;
    vlong n;

    s = 0;
    n = 0;
    do {
        if(p == ep)
            return -1;
        c = *p++;
        n |= (vlong)(c & 0x7f) << s;
        s += 7;
    } while (c & 0x80 && s < 63);
    *pp = p;
    return n;
}

```

17.8.2 Rdelta

```

⟨readpacked() switch object type t cases 179b⟩+≡ (161e) <177c
case GRdelta:
    if(readrdelta(f, o, l, flag) == ERROR_NEG1)
        return ERROR_NEG1;
    break;

```

`<Cxxx other cases 180a>≡ (33c) 180c▷`
`Cthin = 1 << 6,`

`<readidxobject() if Cthin flag 180b>≡ (43b)`
`if(flag & Cthin)`
`flag &= ~Cidx;`

`<Cxxx other cases 180c>+≡ (33c) <180a 248c▷`
`Cidx = 1 << 3,`

`<readidxobject() if Cidx flag 180d>≡ (43b)`
`if(flag & Cidx)`
`return nil;`

Uses `hashcmp()` 161e and `hparse()` 31d.

`<function readrdelta 180e>≡ (277)`
`static errorneg1`
`readrdelta(Biobuf *f, Object *o, int nd, int flag)`
`{`
`Object *b;`
`Hash h;`
`char *d;`
`int n;`

`d = nil;`
`if(Bread(f, h.h, sizeof(h.h)) != sizeof(h.h))`
`goto error;`
`if(hasheq(&o->hash, &h))`
`goto error;`
`if((n = decompress(&d, f, nil)) == -1)`
`goto error;`
`o->len = Boffset(f) - o->off;`
`if(d == nil || n != nd)`
`goto error;`
`if((b = readidxobject(f, h, flag|Cthin)) == nil)`
`goto error;`
`if(applydelta(o, b, d, n) == -1)`
`goto error;`
`free(d);`
`return OK_0;`
`error:`
`free(d);`
`return ERROR_NEG1;`
`}`

Uses `applydelta()` 46e, `clear()` 160c, and `readpacked()` 180e.

Chapter 18

Exchanging Objects

18.1 Upstream and remote origin

```
<init.rc check $upstream flag 181a>≡ (123a)
    if(~ $#upstream 0){
        upstream='{git/conf 'defaults "origin".baseurl}'
        if(! ~ $#upstream 0)
            upstream=$upstream/$name
    }
```

```
<init.rc set origin in .git/config 181b>≡ (123a)
    if(! ~ $#upstream 0){
        echo '[remote "origin"]'
        echo ' url = '$upstream
    }
```

18.2 Client/server architecture

18.2.1 Conn

```
<struct Conn 181c>≡ (270)
    struct Conn {
        // enum<Connxxx>
        int type;

        fdt rfd;
        fdt wfd;

        <Conn capability fields 210a>
        <Conn http fields 242b>
    };
```

```
<enum Connxxx 181d>≡ (270)
    enum {
        ConnGit,
        <Connxxx other cases 242a>
    };
```

18.3 Pulling updates from another repository: git pull

18.3.1 Usage

```
<pull.rc command line processing 182a>≡ (182b)
flagfmt='d:debug, q:quiet, f:fetchonly, u:upstream upstream'
args=''
eval '{auxrc/getflags $*} || exec auxrc/usage
```

18.3.2 pull.rc

```
<git9/pull.rc 182b>≡
#!/bin/rc -e

rfork en
. /lib/git/common.rc
<fn update(pull.rc) 182c>
gitup
<pull.rc command line processing 182a>

if(~ $#upstream 0)
    upstream=origin
remote='${nl}{git/conf 'remote "'$upstream'".url'}
if(~ $#remote 0){
    remote=$upstream
    upstream=THEM
}

update $upstream $remote
if (~ $fetchonly 1)
    exit

local='{git/branch}
remote='{git/branch | subst '^ (refs/)?heads' 'remotes/'$upstream}

# we have local commits, but the remote hasn't changed.
# in this case, we want to keep the local commits untouched.
if(~ '{git/query HEAD $remote @} '{git/query $remote}){
    echo 'up to date' >[1=2]
    exit
}
<pull.rc after update(), if not fetch only, when remote and HEAD diverged 183c>
```

18.3.3 update()

```
<fn update(pull.rc) 182c>≡ (182b)
fn update{
    upstream=$1
    url=$2
    dflag=()
    <update() (pull.rc) set dflag 183b>
    <update() (pull.rc) set heads 183a>
    if(! ~ $#heads 0)
        heads='-h'^$heads
    {git/get $dflag -u $upstream $heads $url >[2=3] || die $status} | awk '
    /~remote refs\/(heads|tags)\//{
        ref=$2
        hash=$3
```

```

        gsub("^refs/heads", "refs/remotes/'$upstream'", ref)
        outfile = ".git/"ref
        print outfile > "/env/outfile"
        system("mkdir -p '$nl{basename -d $outfile}");\
        close("/env/outfile")
        print hash > outfile;
        close(outfile);
    }
    ' |[3] tr '\x0d' '\x0a'
}

<update() (pull.rc) set heads 183a>≡ (182c)
heads='${nl{
    # safe to split in awk, branch names may not
    # have spaces or newlines.
    git/query '${nl{
        cd .git/refs && \
        walk -f -emp heads remotes | \
        sort -rn | \
        awk '{print $2}'
    }
}

<update() (pull.rc) set dflag 183b>≡ (182c)
if(! ~ $#debug 0)
    dflag='-d'

```

18.3.4 merge

```

<pull.rc after update(), if not fetch only, when remote and HEAD diverged 183c>≡ (182b)
# The remote repository and our HEAD have diverged: we
# need to merge.
if(! ~ '{git/query HEAD $remote @}' '{git/query HEAD}'){
    >[1=2]{
        echo ours:      '{git/query HEAD}'
        echo theirs:   '{git/query $remote}'
        echo common:   '{git/query HEAD $remote @}'
        echo git/merge $remote
    }
    exit diverged
}

oldcommit='{git/query $local}'
newcommit='{git/query $remote}'

if(~ $#quiet 0){
    git/log -s -e $oldcommit'..' $newcommit
    echo $remote': ' $oldcommit '=>' $newcommit
}

git/branch -mnb $remote $local
exit ''

```

18.3.5 git/branch -mnb <remote> <local>

18.4 git/get

18.4.1 Usage

<function usage (git9/get.c) 184a>≡ (269)

```
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-dl] [-b br] [-u upstream] remote\n", argv0);
    fprintf(STDERR, "\t-t-b br: only fetch matching branch 'br'\n");
    fprintf(STDERR, "remote: fetch from this repository\n");
    exits("usage");
}
```

<global upstream(get.c) 184b>≡ (269)

```
// -u
char *upstream = "origin";
```

Uses upstream 184b.

<globals heads(get.c) 184c>≡ (269)

```
Hash heads[64];
int nheads;
```

18.4.2 main()

<function main (git9/get.c) 184d>≡ (269)

```
void
main(int argc, char **argv)
{
    char *s;
    Conn c;

    ARGBEGIN{
        <main() (get.c) command line processing 184e>
        default: usage(); break;
    }ARGEND;
    if(argc != 1)
        usage();

    gitinit(nil, 0, nil);

    if(gitconnect(&c, argv[0], "upload") == ERROR_NEG1)
        sysfatal("could not dial %s: %r", argv[0]);
    if(fetchpack(&c) == ERROR_NEG1)
        sysfatal("fetch failed: %r");
    closeconn(&c);
    exits(nil);
}
```

<main() (get.c) command line processing 184e>≡ (184d) 185a▷

```
case 'u': upstream=EARGF(usage()); break;
```

```

⟨main() (get.c) command line processing 185a⟩+≡ (184d) <184e 196b⟩
case 'h':
    s = EARGF(usage());
    if(nheads < nelem(heads))
        if(hparse(&heads[nheads], s) == 0)
            nheads++;
    break;

```

18.4.3 gitconnect() part 1

```

⟨function gitconnect 185b⟩≡ (279)
/// main(get.c) | main(send.c) -> <>
errorneg1
gitconnect(Conn *c, char *uri, char *direction)
{
    char path[Npath];
    ⟨gitconnect() other locals 206a⟩

    memset(c, 0, sizeof(Conn));
    c->rfd = c->wfd = c->cfid = -1;

    if(localrepo(uri, path, sizeof(path)) == OK_0)
        return servelocal(c, path, direction);
    // else
    ⟨gitconnect() when uri not local repo 206b⟩
}

```

Uses dialhgit() 208d and dialhttp().

18.4.4 closeconn()

```

⟨function closeconn 185c⟩≡ (279)
void
closeconn(Conn *c)
{
    close(c->rfd);
    close(c->wfd);
    switch(c->type){
        ⟨closeconn() switch c->type cases 208c⟩
    }
}

```

18.4.5 localrepo() and servelocal()

```

⟨function localrepo 185d⟩≡ (279)
static errorneg1
localrepo(char *uri, char *path, int npath)
{
    fdt fd;

    snprintf(path, npath, "%s/.git/../../", uri);
    fd = open(path, OREAD);
    if(fd < 0)
        return ERROR_NEG1;
    if(fd2path(fd, path, npath) != 0){
        close(fd);
        return ERROR_NEG1;
    }
}

```

```

    close(fd);
    return OK_0;
}

```

Uses `Nhost-35` 279, `Npath-36` 279, `Nport-34` 279, `Nproto-33` 279, `dialssh()` 242g, `localrepo()` 186b, `parseuri()` 206c, and `servelocal()` 207b.

`<function servelocal 186a>`≡ (279)

```

/// gitconnect -> <>
static int
servelocal(Conn *c, char *path, char *direction)
{
    int pid;
    fdt pfd[2];

    if(pipe(pfd) == ERROR_NEG1)
        sysfatal("unable to open pipe: %r");
    pid = fork();
    <servelocal() sanity check pid 186b>
    if(pid == 0){
        // children
        close(pfd[1]);
        dup(pfd[0], 0);
        dup(pfd[0], 1);
        execl("/bin/git/serve", "serve", "-w", nil);
        sysfatal("exec: %r");
    }
    // else parent
    close(pfd[0]);
    c->type = ConnGit;
    c->rfd = pfd[1];
    c->wfd = dup(pfd[1], -1);
    return githandshake(c, nil, path, direction);
}

```

Uses `ConnGit` 249a and `githandshake()` 246b.

`<servelocal() sanity check pid 186b>`≡ (186a)

```

if(pid == -1)
    sysfatal("unable to fork");

```

18.4.6 fetchpack()

`<function fetchpack 186c>`≡ (269)

```

int
fetchpack(Conn *c)
{
    char spinner[] = {'|', '/', '-', '\\'};
    char buf[Pktmax], caps[512], *sp[3], *ep;
    char *packtmp, *idxtmp, **ref, *rp;
    Hash h, *have, *want;
    int nref, refs, first, nsent;
    int i, j, l, n, spin, req, pfd;
    vlong packs;
    Objset hadobj;
    Object *o;
    Objq haveq;
    Qelt e;

    nref = 0;

```

```

refsz = 16;
first = 1;
have = eamalloc(refsz, sizeof(have[0]));
want = eamalloc(refsz, sizeof(want[0]));
ref = eamalloc(refsz, sizeof(ref[0]));
while(true){
    n = readpkt(c, buf, sizeof(buf));
    if(n == -1)
        return -1;
    if(n == 0)
        break;

    if(first && n > strlen(buf)){
        parsecaps(buf + strlen(buf) + 1, c);
        if(c->symfrom[0] != '\0')
            print("symref %s %s\n", c->symfrom, c->symto);
    }
    first = 0;

    if(getfields(buf, sp, nelem(sp), 1, " \t\n\r") < 2)
        sysfatal("invalid ref line");
    if(strstr(sp[1], "^{ }"))
        continue;
    if(!okrefname(sp[1]))
        sysfatal("remote side sent invalid ref: %s", sp[1]);
    if(fetchbranch && !branchmatch(sp[1], fetchbranch))
        continue;
    else if(strcmp(sp[1], "HEAD") != 0
        && !prefixed(sp[1], "refs/heads/")
        && !prefixed(sp[1], "refs/tags/"))
        continue;
    if(refsz == nref + 1){
        refsz *= 2;
        have = earealloc(have, refsz, sizeof(have[0]));
        want = earealloc(want, refsz, sizeof(want[0]));
        ref = earealloc(ref, refsz, sizeof(ref[0]));
    }
    if(hparse(&want[nref], sp[0]) == ERROR_NEG1)
        sysfatal("invalid hash %s", sp[0]);
    if (resolveremote(&have[nref], sp[1]) == -1)
        memset(&have[nref], 0, sizeof(have[nref]));
    ref[nref] = estrdup(sp[1]);
    nref++;
}
if(listonly){
    flushpkt(c);
    goto showrefs;
}
// else

if(writephase(c) == -1)
    sysfatal("write: %r");
req = 0;
fmtcaps(c, caps, sizeof(caps));
for(i = 0; i < nref; i++){
    if(hasheq(&have[i], &want[i]))
        goto skip;
    for(j = 0; j < i; j++)
        if(hasheq(&want[i], &want[j]))
            goto skip;
}

```

```

    if((o = readobject(want[i])) != nil){
        unref(o);
        continue;
    }
    if(fmtpkt(c, "want %H%s\n", want[i], caps) == -1)
        sysfatal("could not send want for %H", want[i]);
    caps[0] = 0;
    req = 1;
skip:;
}
flushpkt(c);

nsent = 0;
qinit(&haveq);
osinit(&hadobj);
/*
 * We know we have these objects, and we want to make sure that
 * they end up at the front of the queue. Send the 'have lines'
 * first, and then enqueue their parents for a second round of
 * sends.
 */
for(i = 0; i < nref; i++){
    if(hasheq(&have[i], &Zhash) || oshas(&hadobj, have[i]))
        continue;
    if((o = readobject(have[i])) == nil)
        sysfatal("missing expected object: %H", have[i]);
    if(fmtpkt(c, "have %H", o->hash) == -1)
        sysfatal("write: %r");
    enqueueparent(&haveq, o);
    osadd(&hadobj, o);
    unref(o);
    nsent++;
}
/*
 * The other branches we have probably make sense to send,
 * since often we'll be pulling a new branch with objects
 * that we already have; it's not entirely clear what we
 * want to do here.
 */
for(i = 0; i < nheads; i++){
    if((o = readobject(heads[i])) == nil)
        sysfatal("missing expected object: %H", have[i]);
    if(fmtpkt(c, "have %H", o->hash) == -1)
        sysfatal("write: %r");
    enqueueparent(&haveq, o);
    osadd(&hadobj, o);
    unref(o);
    nsent++;
}
/*
 * While we could short circuit this and check if upstream has
 * acked our objects, for the first 256 haves, this is simple
 * enough.
 *
 * Also, doing multiple rounds of reference discovery breaks
 * when using smart http.
 */
while(req && qpop(&haveq, &e) && nsent < 256){
    if(oshas(&hadobj, e.o->hash))
        continue;

```

```

    if((o = readobject(e.o->hash)) == nil)
        sysfatal("missing object we should have: %H", e.o->hash);
    if(fmtpkt(c, "have %H", o->hash) == -1)
        sysfatal("write: %r");
    enqueueparent(&haveq, o);
    osadd(&hadobj, o);
    unref(o);
    nsent++;
}
osclear(&hadobj);
qclear(&haveq);
if(!req)
    flushpkt(c);
if(fmtpkt(c, "done\n") == -1)
    sysfatal("write: %r");
if(!req)
    goto showrefs;
if(readphase(c) == -1)
    sysfatal("read: %r");

if((packtmp = smprint(".git/objects/pack/fetch.%d.pack", getpid())) == nil)
    sysfatal("smprint: %r");
if((idxtmp = smprint(".git/objects/pack/fetch.%d.idx", getpid())) == nil)
    sysfatal("smprint: %r");
if(mkoutpath(packtmp) == -1)
    sysfatal("could not create %s: %r", packtmp);
if((pfd = create(packtmp, ORDWR, 0664)) == -1)
    sysfatal("could not create %s: %r", packtmp);

fprintf(2, "fetching... ");
packsz = 0;
if(c->multiack){
    for(i = 0; i < nsent; i++){
        if(readpkt(c, buf, sizeof(buf)) == -1)
            sysfatal("read: %r");
        if(strncmp(buf, "NAK\n", 4) == 0)
            break;
        if(strncmp(buf, "ACK ", 4) == 0){
            if(getfields(buf, sp, nelem(sp), 1, " \t") == 2)
                break;
            continue;
        }
        sysfatal("bad response: '%s'", buf);
    }
}
if(readpkt(c, buf, sizeof(buf)) == -1)
    sysfatal("read: %r");
if(!c->sideband && !c->sideband64k && !c->multiack){
    /*
     * Work around torvalds git bug: we get duplicate have lines
     * sometimes, even though the protocol is supposed to start the
     * pack file immediately.
     *
     * Skip ahead until we read 'PACK' off the wire
     */
    while(true){
        if(readn(c->rfd, buf, 4) != 4)
            sysfatal("fetch packfile: short read");
        if(strncmp(buf, "PACK", 4) == 0)
            break;
    }
}

```

```

    buf[4] = 0;
    l = strtol(buf, &ep, 16);
    if(ep != buf + 4)
        sysfatal("fetch packfile: junk pktline");
    if(readn(c->rfd, buf, l-4) != l-4)
        sysfatal("fetch packfile: short read");
}
if(write(pfd, "PACK", 4) != 4)
    sysfatal("write pack header: %r");
packsz = 4;
}
spin = 0;
while(true){
    n = sbread(c, buf, sizeof buf, &rp);
    if(n == 0)
        break;
    if(n == -1 || write(pfd, rp, n) != n)
        sysfatal("fetch packfile: %r");
    if(interactive && spin++ % 100 == 0)
        fprintf(2, "\b%c", spinner[spin/100 % nelem(spinner)]);
    packsiz += n;
}
fprintf(2, "\n");
closeconn(c);
if(seek(pfd, 0, 0) == -1)
    fail(packtmp, idxtmp, "packfile seek: %r");
if(checkhash(pfd, packsiz, &h) == -1)
    fail(packtmp, idxtmp, "corrupt packfile: %r");
close(pfd);
if(indexpack(packtmp, idxtmp, h) == -1)
    fail(packtmp, idxtmp, "could not index fetched pack: %r");
if(rename(packtmp, idxtmp, h) == -1)
    fail(packtmp, idxtmp, "could not rename indexed pack: %r");

showrefs:
    for(i = 0; i < nref; i++){
        print("remote %s %H local %H\n", ref[i], want[i], have[i]);
        free(ref[i]);
    }
    free(ref);
    free(want);
    free(have);
    return 0;
}

```

Uses Zhash [30b](#), branchmatch() [194a](#), closeconn() [245c](#), eamalloc(), earealloc() [257a](#), enqueueparent() [194d](#), estrdup() [257b](#), fetchbranch, flushpkt() [209a](#), fmtcaps() [195a](#), fmtpkt() [209b](#), hasheq(), heads [184b](#), hparse() [31d](#), indexpack() [52a](#), interactive, listonly, mkoutpath() [193](#), nheads, okrefname() [196c](#), osadd() [37e](#), osclear() [37d](#), oshas() [38d](#), osinit() [37c](#), parsecaps() [210b](#), prefixed() [194a](#), qclear() [39f](#), qinit() [39d](#), qpop(), readobject() [157b](#), readphase() [279](#), readpkt() [209c](#), resolveremote() [190](#), sbread() [195b](#), unref() [34a](#), and writephase() [185b](#).

okrefname()

```

⟨function okrefname 190⟩≡ (269)
/// fetchpack -> <> -> okref
/*
 * Checks the rules for a refname at
 * git/Documentation/protocol-common.txt
 */
bool

```

```

okrefname(char *s)
{
    if(strcmp(s, "HEAD") == 0)
        return true;
    if(strncmp(s, "refs/", 5) == 0)
        return okref(s);
    return false;
}

```

Uses okref() 191.

<function okref 191>≡ (284b)

```

/*
 * Checks the rules for valid ref names, as defined in
 * git/Documentation/protocol-common.txt.
 * It does not check that the ref begins with refs/
 * or is called HEAD, since that's only needed in the
 * clone protocol.
 */
bool
okref(char *ref)
{
    int n, slashed;
    char *p;
    Rune r;

    slashed = 0;
    if(*ref == '/' || *ref == '.')
        return 0;
    for(p = ref; *p != 0; p += n) {
        n = chartorune(&r, p);
        switch(r){
            case '.':
                if(p[1]== 0 || p[1] == '.')
                    return 0;
                if(strcmp(p, ".lock") == 0)
                    return 0;
                break;
            case '/':
                if(p[1] == 0 || p[1] == '.' || p[1] == '/')
                    return 0;
                slashed = 1;
                break;
            case '@':
                if(p[1] == '{')
                    return 0;
                break;
            case ' ':
            case '~':
            case '^':
            case ':':
            case '?':
            case '*':
            case '[':
            case '\\':
            case Runeerror:
            case 0x7f: /* DEL */
                return 0;
            default:
                if(r < 0x20 || isspace(r))
                    return 0;
        }
    }
}

```

```

    }
}
if(ref == p)
    return 0;
return slashed;
}

```

resolveremote()

```

<function resolveremote 192a>≡ (269)
/// fetchpack -> <>
int
resolveremote(Hash *h, char *ref)
{
    char buf[128], *s;
    int r, f;

    ref = strip(ref);
    if((r = hparse(h, ref)) != ERROR_NEG1)
        return r;
    /* Slightly special handling: translate remote refs to local ones. */
    if(strcmp(ref, "HEAD") == 0){
        snprintf(buf, sizeof(buf), ".git/HEAD");
    }else if(strstr(ref, "refs/heads") == ref){
        ref += strlen("refs/heads");
        snprintf(buf, sizeof(buf), ".git/refs/remotes/%s/%s", upstream, ref);
    }else if(strstr(ref, "refs/tags") == ref){
        ref += strlen("refs/tags");
        snprintf(buf, sizeof(buf), ".git/refs/tags/%s/%s", upstream, ref);
    }else{
        return -1;
    }

    r = ERROR_NEG1;
    s = strip(buf);
    if((f = open(s, OREAD)) == -1)
        return -1;
    if(readn(f, buf, sizeof(buf)) >= 40)
        r = hparse(h, buf);
    close(f);

    if(r == -1 && strstr(buf, "ref:") == buf)
        return resolveremote(h, buf + strlen("ref:"));
    return r;
}

```

Uses `hparse()` 31d, `resolveremote()` 190, and `upstream` 184b.

rename()

```

<function rename 192b>≡ (269)
/// fetchpack -> <>
int
rename(char *pack, char *idx, Hash h)
{
    char name[128];
    Dir st;

    nulldir(&st);
}

```

```

st.name = name;
snprintf(name, sizeof(name), "%H.pack", h);
if(access(name, AEXIST) == 0)
    fprintf(2, "warning, pack %s already fetched\n", name);
else if(dirwstat(pack, &st) == -1)
    return -1;
snprintf(name, sizeof(name), "%H.idx", h);
if(access(name, AEXIST) == 0)
    fprintf(2, "warning, pack %s already indexed\n", name);
else if(dirwstat(idx, &st) == -1)
    return -1;
return 0;
}

```

Uses Pktmax.

checkhash()

<function checkhash 193>≡ (269)

```

/// fetchpack -> <>
int
checkhash(fdt fd, vlong sz, Hash *hcomp)
{
    DigestState *st;
    Hash hexpect;
    char buf[Pktmax];
    vlong n, r;
    int nr;

    if(sz < 28){
        werrstr("undersize packfile");
        return -1;
    }

    st = nil;
    n = 0;
    while(n != sz - 20){
        nr = sizeof(buf);
        if(sz - n - 20 < sizeof(buf))
            nr = sz - n - 20;
        r = readn(fd, buf, nr);
        if(r != nr)
            return -1;
        st = sha1((uchar*)buf, nr, nil, st);
        n += r;
    }
    sha1(nil, 0, hcomp->h, st);
    if(readn(fd, hexpect.h, sizeof(hexpect.h)) != sizeof(hexpect.h))
        sysfatal("truncated packfile");
    if(!hasheq(hcomp, &hexpect)){
        werrstr("bad hash: %H != %H", *hcomp, hexpect);
        return -1;
    }
    return 0;
}

```

Uses hasheq().

mkoutpath()

```
<function mkoutpath 194a>≡ (269)
int
mkoutpath(char *path)
{
    char s[128];
    char *p;
    int fd;

    snprintf(s, sizeof(s), "%s", path);
    for(p=strchr(s+1, '/'); p; p=strchr(p+1, '/')){
        *p = 0;
        if(access(s, AEXIST) != 0){
            fd = create(s, OREAD, DMDIR | 0775);
            if(fd == -1)
                return -1;
            close(fd);
        }
        *p = '/';
    }
    return 0;
}
```

Misc helpers

```
<function prefixed 194b>≡ (269)
bool
prefixed(char *s, char *pfx)
{
    return strncmp(s, pfx, strlen(pfx)) == 0;
}
```

Uses prefixed() 194a.

```
<function branchmatch 194c>≡ (269)
/// fetchpack -> <>
int
branchmatch(char *br, char *pat)
{
    char name[128];

    if(prefixed(pat, "refs/heads"))
        snprintf(name, sizeof(name), "%s", pat);
    else if(prefixed(pat, "heads/"))
        snprintf(name, sizeof(name), "refs/%s", pat);
    else
        snprintf(name, sizeof(name), "refs/heads/%s", pat);
    return strcmp(br, name) == 0;
}
```

```
<function fail 194d>≡ (269)
void
fail(char *pack, char *idx, char *msg, ...)
{
    char buf[ERRMAX];
    va_list ap;

    va_start(ap, msg);
    snprintf(buf, sizeof(buf), msg, ap);
}
```

```

    va_end(ap);

    remove(pack);
    remove(idx);
    fprintf(2, "%s", buf);
    exits(buf);
}

```

Uses GCommit 33a and readobject() 157b.

```

⟨function enqueueparent 195a⟩≡ (269)
    /// fetchpack -> <>
    void
    enqueueparent(Objq *q, Object *o)
    {
        Object *p;
        int i;

        if(o->type != GCommit)
            return;
        for(i = 0; i < o->commit->nparent; i++){
            if((p = readobject(o->commit->parent[i])) == nil)
                continue;
            qput(q, p, 0);
            unref(p);
        }
    }

```

fmtcaps()

```

⟨function fmtcaps 195b⟩≡ (269)
    /// fetchpack -> <>
    void
    fmtcaps(Conn *c, char *caps, int ncaps)
    {
        char *p, *e;

        p = caps;
        e = caps + ncaps;
        *p = 0;
        if(c->multiack)
            p = seprint(p, e, " multi_ack");
        if(c->sideband64k)
            p = seprint(p, e, " side-band-64k");
        else if(c->sideband)
            p = seprint(p, e, " side-band");
        assert(p != e);
    }

```

Uses Pktmax and readpkt() 209c.

sbread()

```

⟨function sbread 195c⟩≡ (269)
    /// fetchpack -> <>
    int
    sbread(Conn *c, char *buf, int nbuf, char **pbuf)
    {
        int n;
    }

```

```

assert(nbuf >= Pktmax);
if(!c->sideband && !c->sideband64k){
    *pbuf = buf;
    return readn(c->rfd, buf, nbuf);
}else{
    *pbuf = buf+1;
    while(true){
        n = readpkt(c, buf, nbuf);
        if(n <= 0)
            return n;
        else if(buf[0] == 1 && n > 1)
            return n - 1;
        else if(buf[0] == 3)
            fprintf(2, "error: %s\n", buf+1);
        else if(buf[0] < 1 || buf[0] > 3)
            fprintf(2, "unknown sideband(%c:%d) data: %s\n", buf[0], buf[0], buf+1);
    }
}
}
}

```

Uses Pktmax.

18.4.7 Advanced features

git/get -l, list only

```

<global listonly(get.c 196a)>≡ (269)
    bool listonly;

```

```

<main()(get.c) command line processing 196b>+≡ (184d) <185a 196d>
    case 'l': listonly=true; break;

```

git/get -b, fetch branch

```

<global fetchbranch(get.c 196c)>≡ (269)
    char *fetchbranch;

```

```

<main()(get.c) command line processing 196d>+≡ (184d) <196b 254b>
    case 'b': fetchbranch=EARGF(usage()); break;

```

18.5 Pushing changes to another repository: git push

18.5.1 Usage

```

<push.rc command line processing 196e)>≡ (197a)
    flagfmt='a:pushall, b:branch branch, f:force, d:debug,
        r:remove remove, u:upstream upstream'
    args=''
    eval '{auxrc/getflags $*} || exec auxrc/usage

```

18.5.2 push.rc

<git9/push.rc 197a>≡

```
#!/bin/rc -e

rfork en
. /lib/git/common.rc
gitup
<push.rc command line processing 196e>
if(! ~ $#* 0)
    exec auxrc/usage

if(~ $pushall 1)
    branch='${nl}{cd .git/refs/heads && walk -f}
if(~ $#branch 0)
    branch='{git/branch}
if(~ $#branch 0)
    die 'no branches'

if(~ $force 1)
    force=-f
if(~ $debug 1)
    debug='-d'

if(~ $#upstream 0)
    upstream=origin
remotes='${nl}{git/conf -a 'remote "'$upstream'".url'}
if(~ $#remotes 0)
    remotes=$upstream

branch=-b~$branch
if(! ~ $#remove 0)
    remove=-r~$remove

for(remote in $remotes){
    updates='${nl}{git/send $debug $force $branch $remove $remote} || die $status
    for(ln in $updates){
        <push.rc when iterating remotes and updates line ln 197b>
    }
}
exit ''
```

<push.rc when iterating remotes and updates line ln 197b>≡

(197a)

```
u='{echo $ln}
refpath='{echo $u(2) | subst '~refs/heads/' '.git/refs/remotes/'$upstream''}
switch($u(1)){
case update;
    mkdir -p '{basename -d $refpath}
    echo $u(4) > $refpath
    echo $u(2)~': ' $u(3) '=>' $u(4)
case delete;
    echo $u(2)~': removed'
    rm -f $refpath
case uptodate;
    echo $u(2)~': up to date'
}
```

18.6 git/send

18.6.1 Usage

```
<function usage (git9/send.c) 198a>≡ (283a)
void
usage(void)
{
    fprintf(STDERR, "usage: %s remote [reponame]\n", argv0);
    exits("usage");
}
```

18.6.2 main()

```
<function main (git9/send.c) 198b>≡ (283a)
void
main(int argc, char **argv)
{
    char *br;
    Conn c;

    ARGBEGIN{
        <main() (send.c) command line processing 202b>
        default: usage(); break;
    }ARGEND;
    if(argc != 1)
        usage();

    gitinit(nil, 0, nil);
    if(gitconnect(&c, argv[0], "receive") == ERROR_NEG1)
        sysfatal("git connect: %s: %r", argv[0]);
    if(sendpack(&c) == ERROR_NEG1)
        sysfatal("send failed: %r");
    closeconn(&c);
    exits(nil);
}
```

18.6.3 sendpack()

```
<struct Map 198c>≡ (283a)
struct Map {
    char *ref;
    Hash ours;
    Hash theirs;
};
```

```
<function findkey 198d>≡ (283a)
int
findkey(Map *m, int nm, char *ref)
{
    int i;
    for(i = 0; i < nm; i++)
        if(strcmp(m[i].ref, ref) == 0)
            return i;
    return -1;
}
```

<function sendpack 199>≡

(283a)

```
/// main(send.c) -> <>
int
sendpack(Conn *c)
{
    int i, n, idx, nsp, send;
    bool first;
    int nours, ntheirs, nmap;
    char buf[Pktmax], *sp[3];
    Hash h, *theirs, *ours;
    Object *a, *b, *p, *o;
    char **refs;
    Map *map, *m;

    first = true;
    nours = readours(&ours, &refs);
    theirs = nil;
    ntheirs = 0;
    nmap = nours;
    map = eamalloc(nmap, sizeof(Map));
    for(i = 0; i < nmap; i++){
        map[i].theirs = Zhash;
        map[i].ours = ours[i];
        map[i].ref = refs[i];
    }
    while(true){
        n = readpkt(c, buf, sizeof(buf));
        if(n == -1)
            return -1;
        if(n == 0)
            break;
        if(first && n > strlen(buf))
            parsecaps(buf + strlen(buf) + 1, c);
        first = false;

        if(getfields(buf, sp, nelem(sp), 1, " \t\r\n") != 2)
            sysfatal("invalid ref line %.*s", utfnlen(buf, n), buf);
        theirs = earealloc(theirs, ntheirs+1, sizeof(Hash));
        if(hparse(&theirs[ntheirs], sp[0]) == -1)
            sysfatal("invalid hash %s", sp[0]);
        if((idx = findkey(map, nmap, sp[1])) != -1)
            map[idx].theirs = theirs[ntheirs];
        /*
         * we only keep their ref if we can read the object to add it
         * to our reachability; otherwise, discard it; we only care
         * that we don't have it, so we can tell whether we need to
         * bail out of pushing.
         */
        if((o = readobject(theirs[ntheirs])) != nil){
            ntheirs++;
            unref(o);
        }
    }

    if(writephase(c) == -1)
        return -1;
    send = 0;
    if(force)
        send=1;
    for(i = 0; i < nmap; i++){
```

```

m = &map[i];
a = readobject(m->theirs);
if(hasheq(&m->ours, &Zhash))
    b = nil;
else
    b = readobject(m->ours);
p = nil;
if(a != nil && b != nil)
    p = ancestor(a, b);
if(!force
&& !hasheq(&m->theirs, &Zhash)
&& !hasheq(&m->ours, &Zhash)
&& (a == nil || p != a)){
    fprintf(2, "remote has diverged\n");
    werrstr("remote diverged");
    flushpkt(c);
    return -1;
}
unref(a);
unref(b);
unref(p);
if(hasheq(&m->theirs, &m->ours)){
    print("uptodate %s\n", m->ref);
    continue;
}
print("update %s %H %H\n", m->ref, m->theirs, m->ours);
n = snprintf(buf, sizeof(buf), "%H %H %s", m->theirs, m->ours, m->ref);

/*
 * Workaround for github.
 *
 * Github will accept the pack but fail to update the references
 * if we don't have capabilities advertised. Report-status seems
 * harmless to add, so we add it.
 *
 * Github doesn't advertise any capabilities, so we can't check
 * for compatibility. We just need to add it blindly.
 */
if(i == 0 && c->report){
    buf[n++] = '\0';
    n += snprintf(buf + n, sizeof(buf) - n, " report-status");
}
if(writepkt(c, buf, n) == ERROR_NEG1)
    sysfatal("unable to send update pkt");
send = 1;
}
flushpkt(c);
if(!send){
    fprintf(2, "nothing to send\n");
    return 0;
}

if(writepack(c->wfd, ours, nours, theirs, ntheirs, &h) == -1)
    return -1;
if(!c->report)
    return 0;

if(readphase(c) == -1)
    return -1;
/* We asked for a status report, may as well use it. */

```

```

while((n = readpkt(c, buf, sizeof(buf))) > 0){
    buf[n] = 0;
    if(chattygit)
        fprintf(2, "done sending pack, status %s\n", buf);
    nsp = getfields(buf, sp, nelem(sp), 1, " \t\n\r");
    if(nsp < 2)
        continue;
    if(nsp < 3)
        sp[2] = "";
    /*
     * Only report errors; successes will be reported by
     * surrounding scripts.
     */
    if(strcmp(sp[0], "unpack") == 0 && strcmp(sp[1], "ok") != 0)
        werrstr("unpack %s", sp[1]);
    else if(strcmp(sp[0], "ng") == 0)
        werrstr("failed update: %s %s", sp[1], sp[2]);
    else
        continue;
    return -1;
}
return 0;
}

```

Uses Zhash 30b, ancestor() 94a, chattygit 253c, eamalloc(), earealloc() 257a, findkey() 201a, flushpkt() 209a, force 283a, hasheq(), hparse() 31d, parsecaps() 210b, readobject() 157b, readours() 198d, readphase() 279, readpkt() 209c, unref() 34a, writepack() 171a, writephase() 185b, and writepkt() 208e.

findref() and findkey()

```

⟨function findref 201a⟩≡ (283a)
int
findref(char **r, int nr, char *ref)
{
    int i;
    for(i = 0; i < nr; i++)
        if(strcmp(r[i], ref) == 0)
            return i;
    return -1;
}

```

readours()

```

⟨function readours 201b⟩≡ (283a)
int
readours(Hash **tailp, char ***refp)
{
    int nu, i, idx;
    char *r, *pfx, **ref;
    Hash *tail;

    if(sendall)
        return listrefs(tailp, refp);
    nu = 0;
    tail = eamalloc((nremoved + nbranch), sizeof(Hash));
    ref = eamalloc((nremoved + nbranch), sizeof(char*));
    for(i = 0; i < nbranch; i++){
        ref[nu] = estrdup(branch[i]);
        if(resolveref(&tail[nu], branch[i]) == -1)

```

```

        sysfatal("broken branch %s", branch[i]);
    nu++;
}
for(i = 0; i < nremoved; i++){
    pfx = "refs/heads/";
    if(strstr(removed[i], "heads/") == removed[i])
        pfx = "refs/";
    if(strstr(removed[i], "refs/heads/") == removed[i])
        pfx = "";
    if((r = smprint("%s%s", pfx, removed[i])) == nil)
        sysfatal("smprint: %r");
    if((idx = findref(ref, nu, r)) == -1)
        idx = nu++;
    else
        free(ref[idx]);
    assert(idx < nremoved + nbranch);
    memcpy(&tail[idx], &Zhash, sizeof(Hash));
    ref[idx] = r;
}
dprint(1, "nu: %d\n", nu);
for(i = 0; i < nu; i++)
    dprint(1, "update: %H %s\n", tail[i], ref[i]);
*tailp = tail;
*refp = ref;
return nu;
}

```

Uses Zhash 30b, branch 202e, dprint 270, eamalloc(), estrdup() 257b, findref(), listrefs() 150e, nbranch, nremoved, removed, resolveref(), and sendall.

18.6.4 Advanced features

git/send -f, forcing

```

<global force(send.c) 202a>≡ (283a)
    bool force;

```

```

<main() (send.c) command line processing 202b>≡ (198b) 202d▷
    case 'f':
        force=true;
        break;

```

git/send -r, removing

```

<globals removed(send.c) 202c>≡ (283a)
    char *removed[128];
    int nremoved;

```

```

<main() (send.c) command line processing 202d>+≡ (198b) <202b 203a▷
    case 'r':
        if(nremoved == nelem(removed))
            sysfatal("too many deleted branches");
        removed[nremoved++] = EARGF(usage());
        break;

```

git/send -a, sending all

```

<global sendall(send.c) 202e>≡ (283a)
    bool sendall;

```

```

⟨main() (send.c) command line processing 203a⟩+≡ (198b) <202d 203c⟩
case 'a':
    sendall=true;
    break;

```

git/send -b, sending selected branches

```

⟨globals branch (send.c) 203b⟩≡ (283a)
char **branch;
int nbranch;

```

```

⟨main() (send.c) command line processing 203c⟩+≡ (198b) <203a 254d⟩
case 'b':
    br = EARGF(usage());
    if(strncmp(br, "refs/heads/", strlen("refs/heads/")) == 0)
        br = smprint("%s", br);
    else if(strncmp(br, "heads/", strlen("heads/")) == 0)
        br = smprint("refs/%s", br);
    else
        br = smprint("refs/heads/%s", br);
    branch = erealloc(branch, (nbranch + 1)*sizeof(char*));
    branch[nbranch] = br;
    nbranch++;
    break;

```

18.7 Cloning a remote repository: git clone

18.7.1 Usage

```

⟨clone.rc command line processing 203d⟩≡ (203e)
flagfmt='d:debug, b:branch branch'; args='remote [local]'
eval '{auxrc/getflags $*} || exec auxrc/usage

```

18.7.2 clone.rc

```

⟨git9/clone.rc 203e⟩≡
#!/bin/rc
rfork en
. /lib/git/common.rc

⟨clone.rc command line processing 203d⟩

if(~ $debug 1)
    debug=(-d)

remote=$1
local=$2

if(~ $#remote 0)
    exec auxrc/usage
if(~ $#local 0)
    local='{nl{basename '{nl{echo $remote | sed 's@.*/([/]+)*$@\\1@'} .git}
if(~ $#branch 1)
    branchflag=(-b $branch)

if(test -e $local && ~ '{ls $local | sed 1q | wc -l} 1)

```

```

die 'destination already exists:' $local

fn clone{
    flag +e
    mkdir -p $local/.git
    mkdir -p $local/.git/fs
    mkdir -p $local/.git/objects/pack/
    mkdir -p $local/.git/refs/heads/

    cd $local

    >>.git/config {
        echo '[remote "origin",'
        echo ' url='$remote
    }
    {git/get $debug $branchflag $remote >[2=3] | awk '
    BEGIN{
        headref=""
        if(ENVIRON["branch"] != "")
            headref="refs/remotes/origin/"ENVIRON["branch"]
        headhash=""
    }
    /^symref / && headref == "" {
        if($2 == "HEAD"){
            gsub("^refs/heads", "refs/remotes/origin", $3)
            gsub("^refs/tags", "refs/remotes/origin/tags", $3)
        }
    }
    /^remote /{
        if($2=="HEAD"){
            headhash=$3
        }else if(match($2, "^refs/(heads|tags)/")){
            gsub("^refs/heads", "refs/remotes/origin", $2)
            if($2 == headref || (headref == "" && $3 == headhash))
                headref=$2
            outfile = ".git/" $2
            outdir = outfile
            gsub("/?[~/]*/?$", "", outdir)
            system(sprintf("mkdir -p %q", outdir))
            print $3 > outfile
            close(outfile)
        }
    }
    END{
        if(headref != ""){
            remote = headref;
            refdir = headref;
            gsub("/?[~/]*/?$", "", refdir)
            gsub("^refs/remotes/origin", "refs/heads", headref)
            system("mkdir -p '{basename -d .git/"headref}");
            system("cp .git/" remote " .git/" headref)
            print "ref: " headref > ".git/HEAD"
        }else if(headhash != ""){
            print "warning: detached head "headhash > "/fd/2"
            print headhash > ".git/HEAD"
        }
    }
    } | [3] tr '\x0d' '\x0a' || die 'could not clone repository'

tree=.git/fs/HEAD/tree

```

```

lbranch='{git/branch}
rbranch='{echo $lbranch | subst 'heads' 'remotes/origin'}
echo checking out repository...
if(test -f .git/refs/$rbranch){
    mkdir -p '{basename -d .git/refs/$lbranch}
    cp .git/refs/$rbranch .git/refs/$lbranch
    git/fs
    @ {builtin cd $tree && tar cif /fd/1 .} | @ {tar xf /fd/0} \
        || die 'checkout failed:' $status
    {for(f in '$nl{cd $tree && walk -f})
        echo 'T NOQID 0 '$f} > .git/INDEX9
    }
if not{
    echo no default branch >[1=2]
    echo check out your code with git/branch >[1=2]
}
}

fn sigint {
    echo cancelled clone $remote: cleaning $local >[1=2]
    umount $local/.git/fs >[2]/dev/null
    rm -rf $local
    exit interrupted
}

@{clone}
st=$status
if(! ~ $st ''){
    echo failed to clone $remote: cleaning $local >[1=2]
    umount $local/.git/fs >[2]/dev/null
    rm -rf $local
    exit $st
}
exit ''

```

Chapter 19

Networking

19.1 gitconnect() part 2

```
<gitconnect() other locals 206a>≡ (185b)
    char proto[Nproto], host[Nhost], port[Nport];
```

```
<gitconnect() when uri not local repo 206b>≡ (185b)
    if(parseuri(uri, proto, host, port, path) == ERROR_NEG1){
        werrstr("bad uri %s", uri);
        return ERROR_NEG1;
    }
    if(strcmp(proto, "git") == 0)
        return dialgit(c, host, port, path, direction);
    <gitconnect() when uri not local repo, if series on proto cases 242c>
    // else
    werrstr("unknown protocol %s", proto);
    return ERROR_NEG1;
```

Uses ConnHttp and dprint 270.

```
<function parseuri 206c>≡ (279)
    static int
    parseuri(char *uri, char *proto/*OUT*/, char *host/*OUT*/, char *port/*OUT*/, char *path/*OUT*/)
    {
        char *s, *p, *q;
        bool hasport;

        print("uri: \"%s\"\n", uri);

        p = strstr(uri, "://");
        if(p == nil)
            snprint(proto, Nproto, "ssh");
        else if(strncmp(uri, "git+", 4) == 0)
            grab(proto, Nproto, uri + 4, p);
        else
            grab(proto, Nproto, uri, p);
        *port = '\0';
        hasport = true;
        if(strcmp(proto, "git") == 0)
            snprint(port, Nport, "9418");
        <parseuri() if series on proto cases 242d>
        else
            hasport = false;

        s = (p != nil) ? p + 3 : uri;
        p = nil;
```

```

if(!hasport){
    p = strstr(s, ":");
    if(p != nil)
        p++;
}
if(p == nil)
    p = strstr(s, "/");
if(p == nil || strlen(p) == 1){
    werrstr("missing path");
    return ERROR_NEG1;
}

q = memchr(s, ':', p - s);
if(q){
    grab(host, Nhost, s, q);
    grab(port, Nport, q + 1, p);
}else{
    grab(host, Nhost, s, p);
}

snprint(path, Npath, "%s", p);
return 0;
}

```

Uses Nhost-35 279, Npath-36 279, Nport-34 279, Nproto-33 279, and grab() 207a.

```

⟨function grab 207a⟩≡ (279)
static void
grab(char *dst, int n, char *p, char *e)
{
    int l;

    l = e - p;
    if(l >= n)
        sysfatal("overlong component");
    memcpy(dst, p, l);
    dst[l] = '\0';
}

```

19.2 git:// protocol

19.2.1 dialgit()

```

⟨function dialgit 207b⟩≡ (279)
/// (main(get.c) | main(send.c)) -> gitconnect -> <>
static errorneg1
dialgit(Conn *c, char *host, char *port, char *path, char *direction)
{
    char *ds; // dial string
    fdt fd;

    ds = netmkaddr(host, "tcp", port);
    ⟨dialgit() sanity check ds 208a⟩
    dprint(1, "dial %s git-%s-pack %s\n", ds, direction, path);
    fd = dial(ds, nil, nil, nil);
    ⟨dialgit() sanity check fd 208b⟩
    c->type = ConnGit;
    c->rfd = fd;
    c->wfd = dup(fd, -1);
}

```

```

    return githandshake(c, host, path, direction);
}

```

Uses ConnGit 249a, dprint 270, and githandshake() 246b.

```

⟨dialgit() sanity check ds 208a⟩≡ (207b)
    if(ds == nil)
        return ERROR_NEG1;

```

```

⟨dialgit() sanity check fd 208b⟩≡ (207b)
    if(fd == -1)
        return ERROR_NEG1;

```

```

⟨closeconn() switch c->type cases 208c⟩≡ (185c) 242f▷
    case ConnGit:
        break;

```

19.2.2 githandshake()

```

⟨function githandshake 208d⟩≡ (279)
    /// dialgit | servelocal -> <>
    static errorneg1
    githandshake(Conn *c, char *host, char *path, char *direction)
    {
        char *p, *e, cmd[512];

        p = cmd;
        e = cmd + sizeof(cmd);
        p = seprint(p, e - 1, "git-%s-pack %s", direction, path);
        if(host != nil)
            p = seprint(p + 1, e, "host=%s", host);
        if(writepkt(c, cmd, p - cmd + 1) == -1){
            fprintf(STDERR, "failed to write message\n");
            closeconn(c);
            return ERROR_NEG1;
        }
        return OK_0;
    }

```

Uses closeconn() 245c and writepkt() 208e.

19.3 Packet IO

19.3.1 Writing packets

```

⟨function writepkt 208e⟩≡ (279)
    errorneg1
    writepkt(Conn *c, char *buf, int nbuf)
    {
        char len[5];

        snprintf(len, sizeof(len), "%04x", nbuf + 4);
        if(write(c->wfd, len, 4) != 4)
            return ERROR_NEG1;
        if(write(c->wfd, buf, nbuf) != nbuf)
            return ERROR_NEG1;
        tracepkt(1, "<=w=", buf, nbuf);
    }

```

```

    return OK_0;
}

```

Uses `tracepkt()` 254f.

```

<function flushpkt 209a>≡ (279)
int
flushpkt(Conn *c)
{
    dprint(1, "<=w= 0000\n");
    return write(c->wfd, "0000", 4);
}

```

Uses `dprint` 270.

```

<function fmtpkt 209b>≡ (279)
int
fmtpkt(Conn *c, char *fmt, ...)
{
    char pkt[Pktmax];
    va_list ap;
    int n;

    va_start(ap, fmt);
    n = vsnprint(pkt, sizeof(pkt), fmt, ap);
    n = writepkt(c, pkt, n);
    va_end(ap);
    return n;
}

```

Uses `Pktmax` and `writepkt()` 208e.

19.3.2 Reading packets

```

<function readpkt 209c>≡ (279)
int
readpkt(Conn *c, char *buf, int nbuf)
{
    char len[5];
    char *e;
    int n;

    if(readn(c->rfd, len, 4) != 4){
        werrstr("pktline: short read from transport");
        return -1;
    }
    len[4] = 0;
    n = strtol(len, &e, 16);
    if(n == 0){
        dprint(1, "=r=> 0000\n");
        return 0;
    }
    if(e != len + 4 || n <= 4)
        sysfatal("pktline: bad length '%s'", len);
    n -= 4;
    if(n >= nbuf)
        abort();//sysfatal("pktline: undertime buffer");
    if(readn(c->rfd, buf, n) != n)
        return ERROR_NEG1;
    if(n > 4 && strncmp(buf, "ERR ", 4) == 0){
        if((e = strrchr(buf, '\n')) != nil)

```

```

        *e = '\0';
        werrstr("%s", buf + 4);
        return ERROR_NEG1;
    }
    buf[n] = 0;
    tracepkt(1, "=r=>", buf, n);
    return n;
}

```

Uses `dprint` 270 and `tracepkt()` 254f.

19.4 Capabilities

19.4.1 Conn Capabilities

```

<Conn capability fields 210a>≡ (181c)
char    symfrom[256];
char    symto[256];
char    multiack;
char    sideband;
char    sideband64k;
char    report;

```

19.4.2 parsecaps()

```

<function parsecaps 210b>≡ (279)
void
parsecaps(char *caps, Conn *c)
{
    char *p, *n, *s, *t;

    for(p = caps; p != nil; p = n){
        n = strchr(p, ' ');
        if(n != nil)
            *n++ = 0;
        if(matchcap(p, "report-status", 1) != nil)
            c->report = 1;
        if(matchcap(p, "multi_ack", 1) != nil)
            c->multiack = 1;
        else if(matchcap(p, "side-band", 1) != nil)
            c->sideband = 1;
        else if(matchcap(p, "side-band-64k", 1) != nil)
            c->sideband64k = 1;
        else if((s = matchcap(p, "symref=", 0)) != nil){
            if((t = strchr(s, ':')) == nil)
                continue;
            *t++ = '\0';
            snprintf(c->symfrom, sizeof(c->symfrom), s);
            snprintf(c->symto, sizeof(c->symto), t);
        }
    }
}

```

Uses `matchcap()` 210c.

```

<function matchcap 210c>≡ (279)
char *
matchcap(char *s, char *cap, int full)

```

```

{
    if(strncmp(s, cap, strlen(cap)) == 0)
        if(!full || strlen(s) == strlen(cap))
            return s + strlen(cap);
    return nil;
}

```

19.5 git:// client

19.6 git:// server

19.6.1 Usage

```

<function usage (git9/serve.c) 211a>≡ (283b)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-dw] [-r rel]\n", argv0);
    exits("usage");
}

```

```

<global allowwrite(serve.c) 211b>≡ (283b)
// -w
bool allowwrite;

```

19.6.2 main.c

```

<function main (git9/serve.c) 211c>≡ (283b)
void
main(int argc, char **argv)
{
    char *repo, cmd[32], buf[512];
    Conn c;

    ARGBEGIN{
        <main() (serve.c) command line processing 212a>
        default: usage(); break;
    }ARGEND;

    interactive = false;

    //TODO: can do RFNAMEG|RFNOMNT and a single call?
    if(rfork(RFNAMEG) == -1)
        sysfatal("rfork: %r");
    if(rfork(RFNOMNT) == -1)
        sysfatal("rfork: %r");
    <main() (serve.c) if pathpfx 222c>

    initconn(&c, STDIN, STDOUT);
    if(readpkt(&c, buf, sizeof(buf)) == -1)
        sysfatal("readpkt: %r");

    repo = parsecmd(buf, cmd, sizeof(cmd));
    cleanname(repo);
    if(strncmp(repo, "../", 3) == 0)
        fail(&c, "invalid path %s\n", repo);
}

```

```

// sandboxing! chroot.
if(bind(repo, "/", MREPL) == -1)
    fail(&c, "no such repo: %s", repo);
if(chdir("/") == -1)
    fail(&c, "no such repo");

gitinit(nil, 0, nil);

if(strcmp(cmd, "git-receive-pack") == 0)
    recvpack(&c);
else if(strcmp(cmd, "git-upload-pack") == 0)
    servpack(&c);
else
    fail(&c, "unsupported command '%s'", cmd);
exits(nil);
}

```

<main() (serve.c) command line processing 212a> ≡ (211c) 222b▷

```

case 'w':
    allowwrite=true;
    break;

```

<function initconn 212b> ≡ (279)

```

void
initconn(Conn *c, fdt rd, fdt wr)
{
    c->type = ConnGit;
    c->rfd = rd;
    c->wfd = wr;
}

```

<function parsecmd 212c> ≡ (283b)

```

char*
parsecmd(char *buf, char *cmd, int ncmd)
{
    int i;
    char *p;

    for(p = buf, i = 0; *p && i < ncmd - 1; i++, p++){
        if(*p == ' ' || *p == '\t'){
            cmd[i] = 0;
            break;
        }
        cmd[i] = *p;
    }
    while(*p == ' ' || *p == '\t')
        p++;
    return p;
}

```

<function fail (git9/serve.c) 212d> ≡ (283b)

```

_Noreturn static void
fail(Conn *c, char *fmt, ...)
{
    char msg[ERRMAX];
    va_list ap;

    va_start(ap, fmt);
    vsnprint(msg, sizeof(msg), fmt, ap);

```

```

    va_end(ap);
    fmtpkt(c, "ERR %s\n", msg);
    sysfatal("%s", msg);
}

```

19.6.3 git-upload-pack

```

<function servpack 213a>≡ (283b)
    int
    servpack(Conn *c)
    {
        Hash *head, *tail, h;
        int nhead, ntail;

        dprint(1, "negotiating pack\n");
        if(servnegotiate(c, &head, &nhead, &tail, &ntail) == -1)
            fail(c, "negotiate: %r");
        dprint(1, "writing pack\n");
        if(writepack(c->wfd, head, nhead, tail, ntail, &h) == -1)
            fail(c, "send: %r");
        return 0;
    }

```

Uses `dprint` 270, `servnegotiate()` 214, and `writepack()` 171a.

servnegociate()

```

<function servnegotiate 213b>≡ (283b)
    int
    servnegotiate(Conn *c, Hash **head, int *nhead, Hash **tail, int *ntail)
    {
        char pkt[Pktmax];
        int n, acked;
        Object *o;
        Hash h;

        if(showrefs(c) == -1)
            return -1;

        *head = nil;
        *tail = nil;
        *nhead = 0;
        *ntail = 0;
        while(1){
            if((n = readpkt(c, pkt, sizeof(pkt))) == -1)
                goto error;
            if(n == 0)
                break;
            if(strncmp(pkt, "want ", 5) != 0){
                werrstr(" protocol garble %s", pkt);
                goto error;
            }
            if(hparse(&h, &pkt[5]) == -1){
                werrstr(" garbled want");
                goto error;
            }
            if((o = readobject(h)) == nil){
                werrstr("requested nonexistent object");
                goto error;
            }
        }
    }

```

```

    }
    unref(o);
    *head = erealloc(*head, (*nhead + 1)*sizeof(Hash));
    (*head)[*nhead] = h;
    *nhead += 1;
}

acked = 0;
while(1){
    if((n = readpkt(c, pkt, sizeof(pkt))) == -1)
        goto error;
    if(strncmp(pkt, "done", 4) == 0)
        break;
    if(n == 0){
        if(!acked && fmpkt(c, "NAK") == -1)
            goto error;
    }
    if(strncmp(pkt, "have ", 5) != 0){
        werrstr(" protocol garble %s", pkt);
        goto error;
    }
    if(hparse(&h, &pkt[5]) == -1){
        werrstr(" garbled have");
        goto error;
    }
    if((o = readobject(h)) == nil)
        continue;
    if(!acked){
        if(fmpkt(c, "ACK %H", h) == -1)
            goto error;
        acked = 1;
    }
    unref(o);
    *tail = erealloc(*tail, (*ntail + 1)*sizeof(Hash));
    (*tail)[*ntail] = h;
    *ntail += 1;
}
if(!acked && fmpkt(c, "NAK\n") == -1)
    goto error;
return 0;
error:
    fmpkt(c, "ERR %r\n");
    free(*head);
    free(*tail);
    return -1;
}

```

Uses fmpkt() 209b, hparse() 31d, readobject() 157b, readpkt() 209c, showrefs() 215a, and unref() 34a.

showrefs()

```

⟨function showrefs 214⟩≡ (283b)
int
showrefs(Conn *c)
{
    char **names, *s, buf[256];
    int i, ret, nrefs;
    Hash head, *refs;

    ret = -1;

```

```

nrefs = 0;
refs = nil;
names = nil;

if((s = gethead(&head, buf, sizeof(buf))) == nil)
    memset(&head, 0, sizeof(Hash));
if(fmtpkt(c, "%H HEAD%symref=HEAD:%s no-thin\n", head, 0, s) == -1)
    goto error;
if((nrefs = listrefs(&refs, &names)) == -1)
    fail(c, "listrefs: %x");
for(i = 0; i < nrefs; i++){
    if(strncmp(names[i], "heads/", strlen("heads/")) != 0)
        continue;
    if(fmtpkt(c, "%H refs/%s\n", refs[i], names[i]) == -1)
        goto error;
}
if(flushpkt(c) == -1)
    goto error;
ret = 0;
error:
for(i = 0; i < nrefs; i++)
    free(names[i]);
free(names);
free(refs);
return ret;
}

```

Uses flushpkt() 209a, fmtpkt() 209b, gethead() 212d, and listrefs() 150e.

<function gethead 215a>≡ (283b)

```

char*
gethead(Hash *h, char *ref, int nref)
{
    int fd, n;
    char *s;

    if((fd = open(".git/HEAD", OREAD)) == -1)
        return nil;
    if((n = readn(fd, ref, nref-1)) == -1)
        return nil;
    ref[n] = 0;
    strip(ref);
    if(strncmp(ref, "ref: ", 5) != 0)
        return nil;
    s = ref+5;
    if(resolveref(h, s) == -1)
        return nil;
    return s;
}

```

Uses resolveref().

19.6.4 git-receive-pack

<function recvpack 215b>≡ (283b)

```

void
recvpack(Conn *c)
{
    Hash *cur, *upd;
    char **ref;
}

```

```

int nupd;

if(!allowwrite)
    fail(c, "read-only repo");
if(recvnegotiate(c, &cur, &nupd, &ref, &nupd) == -1)
    fail(c, "negotiate refs: %r");
if(nupd != 0 && updatepack(c) == -1)
    sysfatal("update pack: %r");
if(nupd != 0 && updaterefs(c, cur, upd, ref, nupd) == -1)
    sysfatal("update refs: %r");
return;
}

```

Uses allowwrite [222a](#), recvnegotiate() [217a](#), updatepack() [283b](#), and updaterefs().

recvnegociate()

<function recvnegotiate [216](#))≡ ([283b](#))

```

int
recvnegotiate(Conn *c, Hash **cur, Hash **upd, char ***ref, int *nupd)
{
    char pkt[Pktmax], refpath[512], *sp[4];
    Hash old, new;
    int n, i;

    if(showrefs(c) == -1)
        return -1;
    *cur = nil;
    *upd = nil;
    *ref = nil;
    *nupd = 0;
    while(1){
        if((n = readpkt(c, pkt, sizeof(pkt))) == -1)
            goto error;
        if(n == 0)
            break;
        if(getfields(pkt, sp, nelem(sp), 1, " \t\n\r") != 3){
            fmtpkt(c, "ERR protocol garble %s\n", pkt);
            goto error;
        }
        if(hparse(&old, sp[0]) == -1){
            fmtpkt(c, "ERR bad old hash %s\n", sp[0]);
            goto error;
        }
        if(hparse(&new, sp[1]) == -1){
            fmtpkt(c, "ERR bad new hash %s\n", sp[1]);
            goto error;
        }
        if(!validref(sp[2])){
            fmtpkt(c, "ERR invalid ref %s\n", sp[2]);
            goto error;
        }
        *cur = erealloc(*cur, (*nupd + 1)*sizeof(Hash));
        *upd = erealloc(*upd, (*nupd + 1)*sizeof(Hash));
        *ref = erealloc(*ref, (*nupd + 1)*sizeof(Hash));
        (*cur)[*nupd] = old;
        (*upd)[*nupd] = new;
        (*ref)[*nupd] = estrdup(sp[2]);
        n = snprintf(refpath, sizeof(refpath), ".git/%s", sp[2]);
        if(n >= sizeof(refpath)-1){

```

```

        fntpkt(c, "ERR invalid ref %s\n", sp[2]);
        goto error;
    }
    if(access(refpath, AWRITE) == -1
    && access(refpath, AEXIST) == 0){
        fntpkt(c, "ERR read-only ref %s\n", sp[2]);
        goto error;
    }
    *nupd += 1;
}
return 0;
error:
    free(*cur);
    free(*upd);
    for(i = 0; i < *nupd; i++)
        free((*ref)[i]);
    free(*ref);
    return -1;
}

```

Uses `estrdup()` 257b, `fntpkt()` 209b, `hparse()` 31d, `readpkt()` 209c, `showrefs()` 215a, and `validref()` 213a.

```

<function validref 217a>≡ (283b)
    int
    validref(char *s)
    {
        cleannname(s);
        if(strncmp(s, "refs/", 5) != 0)
            return 0;
        return okref(s);
    }

```

Uses `okref()` 191.

updatepack()

```

<function updatepack 217b>≡ (283b)
    int
    updatepack(Conn *c)
    {
        char buf[Pktmax], packtmp[128], idxtmp[128], ebuf[ERRMAX];
        int n, pfd, packsiz;
        Hash h;

        /* make sure the needed dirs exist */
        if(mkdir(".git/objects") == -1)
            return -1;
        if(mkdir(".git/objects/pack") == -1)
            return -1;
        if(mkdir(".git/refs") == -1)
            return -1;
        if(mkdir(".git/refs/heads") == -1)
            return -1;
        snprintf(packtmp, sizeof(packtmp), ".git/objects/pack/recv-%d.pack.tmp", getpid());
        snprintf(idxtmp, sizeof(idxtmp), ".git/objects/pack/recv-%d.idx.tmp", getpid());
        if((pfd = create(packtmp, ORDWR, 0644)) == -1)
            return -1;
        packsiz = 0;
        while(1){
            n = read(c->rfd, buf, sizeof(buf));
            if(n == 0)

```

```

        break;
    if(n == -1){
        rerrstr(ebuf, sizeof(ebuf));
        if(strstr(ebuf, "hungup") == nil)
            return -1;
        break;
    }
    if(write(pfd, buf, n) != n)
        return -1;
    packsz += n;
}
if(checkhash(pfd, packsz, &h) == -1){
    dprint(1, "hash mismatch\n");
    goto error_1;
}
if(indexpack(packtmp, idxtmp, h) == -1){
    dprint(1, "indexing failed: %r\n");
    goto error_1;
}
if(rename(packtmp, idxtmp, h) == -1){
    dprint(1, "rename failed: %r\n");
    goto error_2;
}
return 0;

error_2:  remove(idxtmp);
error_1:  remove(packtmp);
return -1;
}

```

Uses Pktmax, dprint 270, indexpack() 52a, and mkdir().

<function mkdir 218a>≡ (283b)

```

int
mkdir(char *dir)
{
    char buf[ERRMAX];
    int f;

    if(access(dir, AEXIST) == 0)
        return 0;
    if((f = create(dir, OREAD, DMDIR | 0755)) == -1){
        rerrstr(buf, sizeof(buf));
        if(strstr(buf, "exist") == nil)
            return -1;
    }
    close(f);
    return 0;
}

```

<function rename (git9/serve.c) 218b>≡ (283b)

```

int
rename(char *pack, char *idx, Hash h)
{
    char name[128], path[196];
    Dir st;

    nulldir(&st);
    st.name = name;
    snprintf(name, sizeof(name), "%H.pack", h);
    snprintf(path, sizeof(path), ".git/objects/pack/%s", name);
}

```

```

if(access(path, AEXIST) == 0)
    fprintf(2, "warning, pack %s already pushed\n", name);
else if(dirwstat(pack, &st) == -1)
    return -1;
snprint(name, sizeof(name), "%H.idx", h);
snprint(path, sizeof(path), ".git/objects/pack/%s", name);
if(access(path, AEXIST) == 0)
    fprintf(2, "warning, pack %s already indexed\n", name);
else if(dirwstat(idx, &st) == -1)
    return -1;
return 0;
}

```

<function checkhash (git9/serve.c) 219a>≡ (283b)

```

int
checkhash(int fd, vlong sz, Hash *hcomp)
{
    DigestState *st;
    Hash hexpect;
    char buf[Pktmax];
    vlong n, r;
    int nr;

    if(sz < 28){
        werrstr("undersize packfile");
        return -1;
    }

    st = nil;
    n = 0;
    if(seek(fd, 0, 0) == -1)
        sysfatal("packfile seek: %r");
    while(n != sz - 20){
        nr = sizeof(buf);
        if(sz - n - 20 < sizeof(buf))
            nr = sz - n - 20;
        r = readn(fd, buf, nr);
        if(r != nr){
            werrstr("short read");
            return -1;
        }
        st = sha1((uchar*)buf, nr, nil, st);
        n += r;
    }
    sha1(nil, 0, hcomp->h, st);
    if(readn(fd, hexpect.h, sizeof(hexpect.h)) != sizeof(hexpect.h))
        sysfatal("truncated packfile");
    if(!hasheq(hcomp, &hexpect)){
        werrstr("bad hash: %H != %H", *hcomp, hexpect);
        return -1;
    }
    return 0;
}

```

Uses Pktmax and hasheq().

updaterefs() and locking

<function updaterefs 219b>≡ (283b)

```

int

```

```

updaterefs(Conn *c, Hash *cur, Hash *upd, char **ref, int nupd)
{
    char refpath[512], buf[128];
    int i, newidx, hadref, fd, ret;
    fdt lockfd;
    vlong newtm;
    Object *o;
    Hash h;

    ret = -1;
    hadref = 0;
    newidx = -1;
    /*
     * Date of Magna Carta.
     * Wrong because it was computed using
     * the proleptic gregorian calendar.
     */
    newtm = -23811206400;
    if((lockfd = lockrepo()) == -1){
        snprintf(buf, sizeof(buf), "repo locked\n");
        return -1;
    }
    for(i = 0; i < nupd; i++){
        if(resolveref(&h, ref[i]) == 0){
            hadref = 1;
            if(!hasheq(&h, &cur[i])){
                snprintf(buf, sizeof(buf), "old ref changed: %s", ref[i]);
                goto error;
            }
        }
        if(snprintf(refpath, sizeof(refpath), ".git/%s", ref[i]) == sizeof(refpath)){
            snprintf(buf, sizeof(buf), "ref path too long: %s", ref[i]);
            goto error;
        }
        if(hasheq(&upd[i], &Zhash)){
            remove(refpath);
            continue;
        }
        if((o = readobject(upd[i])) == nil){
            snprintf(buf, sizeof(buf), "update to nonexistent hash %H", upd[i]);
            goto error;
        }
        if(o->type != GCommit){
            snprintf(buf, sizeof(buf), "not commit: %H", upd[i]);
            goto error;
        }
        if(o->commit->mtime > newtm){
            newtm = o->commit->mtime;
            newidx = i;
        }
        unref(o);
        if(mkpath(refpath) == -1){
            snprintf(buf, sizeof(buf), "create path: %r");
            goto error;
        }
        if((fd = create(refpath, OWRITE|OTRUNC, 0644)) == -1){
            snprintf(buf, sizeof(buf), "open ref: %r");
            goto error;
        }
        if(fprint(fd, "%H", upd[i]) == -1){

```

```

        snprintf(buf, sizeof(buf), "update ref: %r");
        close(fd);
        goto error;
    }
    close(fd);
}
/*
 * Heuristic:
 * If there are no valid refs, and HEAD is invalid, then
 * pick the ref with the newest commits as the default
 * branch.
 *
 * Several people have been caught out by pushing to
 * a repo where HEAD named differently from what got
 * pushed, and this is going to be more of a footgun
 * when 'master', 'main', and 'front' are all in active
 * use. This should make us pick a useful default in
 * those cases, instead of silently failing.
 */
if(resolveref(&h, "HEAD") == -1 && hadref == 0 && newidx != -1){
    if((fd = create(".git/HEAD", OWRITE|OTRUNC, 0644)) == -1){
        snprintf(buf, sizeof(buf), "open HEAD: %r");
        goto error;
    }
    if(fprintf(fd, "ref: %s", ref[newidx]) == -1){
        snprintf(buf, sizeof(buf), "write HEAD ref: %r");
        goto error;
    }
    close(fd);
}
ret = 0;
error:
    if(ret != 0)
        fmtpkt(c, "ERR %s", buf);
    close(lockfd);
    werrstr(buf);
    return ret;
}

```

Uses GCommit 33a, Zhash 30b, fmtpkt() 209b, hasheq(), lockrepo() 283b, readobject() 157b, resolveref(), and unref() 34a.

```

⟨function lockrepo 221a⟩≡ (283b)
    fdt
    lockrepo(void)
    {
        fdt fd;
        int i;

        for(i = 0; i < 10; i++) {
            if((fd = create(".git/_lock", ORCLOSE|ORDWR|OTRUNC|OEXCL, 0644))!= -1)
                return fd;
            sleep(250);
        }
        return ERROR_NEG1;
    }

```

```

⟨function mkpath 221b⟩≡ (283b)
    int
    mkpath(char *path)
    {
        char *p;

```

```

p = path;
/*
 * we assume that the path has been
 * checked with validref(), so there
 * are no double '/' or odd characters.
 */
while(1){
    p = strchr(p, '/');
    if(p == nil)
        return 0;
    *p = 0;
    if(mkdir(path) == -1)
        return -1;
    *p++ = '/';
}
}
Uses mkdir().

```

19.6.5 Advanced features

git/serve -r <prefix>

```

<global pathpfx(serve.c) 222a>≡ (283b)
char *pathpfx = nil;

```

```

<main() (serve.c) command line processing 222b>+≡ (211c) <212a 254e>
case 'r':
    pathpfx = EARGF(usage());
    if(*pathpfx != '/')
        sysfatal("path prefix must begin with '/'");
    break;

```

```

<main() (serve.c) if pathpfx 222c>≡ (211c)
if(pathpfx != nil){
    if(bind(pathpfx, "/", MREPL) == -1)
        sysfatal("bind: %r");
}

```

Part III
Beyond the Basics

Chapter 20

Core Algorithms

20.1 SHA1

20.2 Murmurhash2

```
<function murmurhash2 224>≡ (284b)
  /// writepack -> readmeta -> loadcommit -> loadtree -> <>
  u64int
  murmurhash2(void *pp, usize n)
  {
    u32int m = 0x5bd1e995;
    u32int r = 24;
    u32int h, k;
    uchar *w, *e;

    h = Seed ^ n;
    e = pp;
    e += n & -4;
    for (w = pp; w != e; w += 4) {
      k = (u32int)w[0] | (u32int)w[1] << 8 | (u32int)w[2] << 16 | (u32int)w[3] << 24;
      k *= m;
      k ^= k >> r;
      k *= m;

      h *= m;
      h ^= k;
    }

    switch (n & 0x3) {
    case 3: h ^= w[2] << 16;
    case 2: h ^= w[1] << 8;
    case 1: h ^= w[0] << 0;
      h *= m;
    }

    h ^= h >> 13;
    h *= m;
    h ^= h >> 15;

    return h;
  }
}
```

20.3 Zip and Unzip

20.4 Diff

20.4.1 Overview

20.4.2 Data structures

20.4.3 Entry point: `Diffs.diff()`

20.4.4 Edit distance basic algorithm ($O(n^2)$ in space/time)

20.5 Diff3

20.5.1 Overview

20.5.2 Data structures

20.5.3 Entry point: `diff3()`

20.5.4 Displaying merging conflicts

Chapter 21

Supporting Programs

21.1 diff

21.1.1 Usage

```
<function usage(diff) 226a>≡ (287a)
void
usage(void)
{
    fprintf(STDERR, "usage: %s [-abcefmmrw] file1 ... file2\n", argv0);
    exits("usage");
}
```

21.1.2 Data structures

21.1.3 main()

```
<function main(diff) 226b>≡ (287a)
void
main(int argc, char *argv[])
{
    int i;
    Dir *fsb, *tsb;

    Binit(&stdout, 1, OWRITE);
    ARGBEGIN{
    case 'e':
    case 'f':
    case 'n':
    case 'c':
    case 'a':
    case 'u':
        mode = ARGC();
        break;
    case 'w':
        bflag = 2;
        break;

    case 'b':
        bflag = 1;
        break;

    case 'r':
        rflag = 1;
```

```

        break;

    case 'm':
        mflag = 1;
        break;

    case 'h':
    default:
        usage();
}ARGEND;

if (argc < 2)
    usage();

if ((tsb = dirstat(argv[argc-1])) == nil)
    sysfatal("can't stat %s", argv[argc-1]);
if (argc > 2) {
    if (!DIRECTORY(tsb))
        sysfatal("not directory: %s", argv[argc-1]);
    mflag = 1;
} else {
    if ((fsb = dirstat(argv[0])) == nil)
        sysfatal("can't stat %s", argv[0]);
    if (DIRECTORY(fsb) && DIRECTORY(tsb))
        mflag = 1;
    free(fsb);
}
free(tsb);
for (i = 0; i < argc-1; i++)
    diff(argv[i], argv[argc-1], 0);

done(anychange);
/*NOTREACHED*/
}

```

<function done(diff) 227a>≡ (287a)

```

void
done(int status)
{
    switch(status)
    {
    case 0:
        exits("");
    case 1:
        exits("some");
    default:
        exits("error");
    }
}

```

21.1.4 Advanced features

21.2 patch

21.2.1 Usage

<function usage(patch.c) 227b>≡ (323b)

```

void

```

```
usage(void)
{
    fprintf(STDERR, "usage: %s [-nR] [-p nstrip] [-r rejfile] [patch...]\n", argv0);
    exits("usage");
}
```

21.2.2 Data structures

21.2.3 main()

(function main(patch.c) 228) ≡ (323b)

```
void
main(int argc, char **argv)
{
    Biobuf *f;
    Patch *p;
    int i, ok;

    ARGBEGIN{
    case 'd':
        workdir = EARGF(usage());
        break;
    case 'r':
        rejfile = EARGF(usage());
        break;
    case 'p':
        strip = atoi(EARGF(usage()));
        break;
    case 'n':
        dryrun++;
        break;
    case 'R':
        reverse++;
        break;
    default:
        usage();
        break;
    }ARGEND;

    ok = 1;
    if(rejfile != nil){
        rejfd = create(rejfile, OWRITE, 0644);
        if(rejfd == -1)
            fail("open %s: %r", rejfile);
    }
    if(argc == 0){
        if((f = Bfdopen(0, OREAD)) == nil)
            fail("open stdin: %r");
        if((p = parse(f, "stdin")) == nil)
            fail("parse patch: %r");
        if(workdir != nil && chdir(workdir) == -1)
            fail("chdir %s: %r", workdir);
        if(apply(p, "stdin") == -1){
            fprintf(2, "apply stdin: %r\n");
            ok = 0;
        }
        freepatch(p);
        Bterm(f);
        ok = finish(ok);
    }
```

```

}else{
    for(i = 0; ok && i < argc; i++){
        if((f = Bopen(argv[i], OREAD)) == nil)
            fail("open %s: %r", argv[i]);
        if((p = parse(f, argv[i])) == nil)
            fail("parse patch: %r");
        if(workdir != nil && chdir(workdir) == -1)
            fail("chdir %s: %r", workdir);
        if(apply(p, argv[i]) == -1){
            fprintf(2, "apply %s: %r\n", argv[i]);
            ok = 0;
        }
        freepatch(p);
        Bterm(f);
        ok = finish(ok);
    }
}
exits(ok ? nil : "failed");
}

```

21.2.4 Advanced features

21.3 merge3

21.3.1 Usage

```

<function usage (diff/merge3.c) (diff) 229a>≡ (310a)
void
usage(void)
{
    fprintf(STDERR, "usage: %s theirs base ours\n", argv0);
    exits("usage");
}

```

21.3.2 Data structures

21.3.3 main()

```

<function main (diff/merge3.c) (diff) 229b>≡ (310a)
void
main(int argc, char **argv)
{
    Diff l, r;
    char *x;

    ARGBEGIN{
    default:
        usage();
    }ARGEND;

    if(argc != 3)
        usage();

    Binit(&stdout, 1, OWRITE);
    memset(&l, 0, sizeof(l));
    memset(&r, 0, sizeof(r));
}

```

```
calcdiff(&l, argv[1], argv[1], argv[0], argv[0]);
calcdiff(&r, argv[1], argv[1], argv[2], argv[2]);

if(l.binary || r.binary)
    sysfatal("cannot merge binaries");

x = merge(&l, &r);

freediff(&l);
freediff(&r);
exits(x);
}
```

Chapter 22

Exchanging Patches

22.1 git/export

22.1.1 Usage

```
<export.rc command line processing 231a>≡ (231b)
  flagfmt='o:patchdir patchdir'; args='[query]'
  eval '{auxrc/getflags $*} || exec auxrc/usage
```

22.1.2 export.rc

```
<git9/export.rc 231b>≡
#!/bin/rc
rfork ne
. /lib/git/common.rc

patchname=/tmp/git.patchname.$pid
patchfile=/tmp/git.patchfile.$pid
fn sigexit{
    rm -f $patchname $patchfile
}

gitup

<export.rc command line processing 231a>

if(~ $#patchdir 1 && ! test -d $patchdir)
    mkdir -p $patchdir

q=$*
if(~ $#q 0)
    q=HEAD
commits='{git/query $q || die $status}'
n=1
m=$#commits

# sleazy hack: we want to run
# under rfork m for the web ui,
# so don't error if we can't mount
mntgen /mnt/scratch >[2]/dev/null || status=''
for(c in $commits){
    cp='{git/query -p $c}'
    pp='{git/query -p $c}'}
```


22.2 git/import

22.2.1 Usage

```
<import.rc command line processing 233a>≡ (233b)
  flagfmt='n:nocommit'; args='file ...'
  eval '{auxrc/getflags $*} || exec auxrc/usage
```

22.2.2 import.rc

```
<git9/import.rc 233b>≡
#!/bin/rc
rfork ne
. /lib/git/common.rc

diffpath=/tmp/gitimport.$pid.diff
fn sigexit {
    rm -f $diffpath
}

fn apply1 {
    adate='{seconds $adate}
    files='${nl}{patch -np1 < $diffpath}
    if(! git/walk -q $files){
        >[1=2] {
            echo patch would clobber files:
            git/walk $files
            exit clobber
        }
    }
    echo applying $msg | sed 1q
    if(! files='${nl}{patch -p1 < $diffpath})
        die 'patch failed'
    for(f in $files){
        if(test -e $f)
            git/add $f
        if not
            git/add -r $f
    }
    git/walk -fRMA $files
    if(~ $#nocommit 0){
        if(hash='{git/save -n $aname -e $amail -N $name -E $email -m $msg -d $adate $parents $files}'){
            echo $hash > $refpath
            for(f in $files)
                echo T NOQID 0 $f >> .git/INDEX9
        }
    }
    exit ''
}

fn apply @{
    git/fs
    amail=''
    aname=''
    msg=''
    whoami
    parents='-p'~'{git/query HEAD}
    branch='{git/branch}
```

```

if(test -e $gitfs/branch/$branch/tree)
    refpath=.git/refs/$branch
if not if(test -e $gitfs/object/$branch/tree)
    refpath=.git/HEAD
if not
    die 'invalid branch:' $branch
awk '
function doapply(){
    if(aname == "" || aemail == "" || date == "" || gotmsg == "")
        exit("missing headers");
    printf "%s", aname > "/env/aname"
    printf "%s", aemail > "/env/aemail"
    printf "%s", date > "/env/adate"
    if(system("rc -c apply1") != 0)
        exit("patch failed");
    close("/env/aname")
    close("/env/aemail")
    close("/env/adate")
    close(ENVIRON["diffpath"])
    applied = 1
}
BEGIN{
    state="headers"
}
state=="diff" && !/^[-+ @]|^diff|^$|^\\ No newline at end of file$/ {
    state="footers";
    doapply();
}
state=="footers" && /@$/ {
    state="headers"
    next
}
state=="headers" && /^[fF]rom:/ {
    sub(/^From:[ \t]*/ , "", $0);
    aname=$0;
    aemail=$0;
    sub(/[ \t]*<.*$/, "", aname);
    sub(/^<[ \t]*</ , "", aemail);
    sub(>[ \t]*>*/ , "", aemail);
}
state=="headers" && /^[Dd]ate:/{
    sub(/^Date:[ \t]*/ , "", $0)
    date=$0
}
state=="headers" && /^Subject:/{
    sub(/^Subject:[ \t]*([Rr][Ee]:)?[ \t]*(\[^\]]*\)[ \t]*)*/ , "", $0);
    gotmsg = 1
    print > "/env/msg"
}
state=="headers" && /^$/ {
    state="body"
}
(state=="headers" || state=="body") && (/^diff / || /^---( |$)/){
    state="diff"
    next
}
state=="body" && /^[ \t]*$/ {
    empty=1
    next
}
}

```

```

state=="body" {
    if(empty)
        printf "\n" > "/env/msg"
    empty=0
    sub(/[ ]+$/, "")
    print > "/env/msg"
}
state=="diff" {
    print > ENVIRON["diffpath"]
}
END{
    if(!failed && state == "diff" && doapply() != 0){
        print "unable to apply patch" > "/fd/2"
        exit("mismatch")
    }
}
,
}

gitup

<import.rc command line processing 233a>

patches=(/fd/0)
if(! ~ $#* 0)
    patches='{cleanname -d $gitrel $*}'
for(p in $patches){
    # upas serves the decoded header and body separately,
    # so we cat them together when applying a upas message.
    #
    # this allows mime-encoded or line-wrapped patches.
    if(test -d $p && test -f $p/header && test -f $p/body)
        {{cat $p/header; echo; cat $p/body} | apply} || die $status
    if not
        apply < $p
}
exit ''

```

Chapter 23

Advanced Features

23.1 Tracking symbolic links

`<Dirent other fields 236a>≡ (34f) 237a▷`
bool_byte islink;

`<parsetree() if symlink 236b>≡ (47c)`
if(m == 0120000){ /* symlink */
t->mode = 0;
t->islink = true;
}

Uses Cidx 33c and Cthin 180a.

`<gitmode() if symlink 236c>≡ (58a)`
if(e->islink)
return 0120000;

`<gtreegen() if e->tree->ent[i].islink 236d>≡ (79b)`
if(e->tree->ent[i].islink)
if((l = walklink(aux, o->data, o->size, 0, &m)) != nil)
o = l;

Uses walklink().

`<objwalk1() when GTree case, if o->tree->ent[i].islink 236e>≡ (79e)`
if(w && o->tree->ent[i].islink)
if((l = walklink(aux, w->data, w->size, 1, &m)) != nil)
w = l;

`<function walklink 236f>≡ (267)`
static Object*
walklink(Gitaux *aux, char *link, int nlink, int ndotdot, int *mode)
{
char *p, *e, *path;
Object *o, *n;
int i;

path = emalloc(nlink + 1);
memcpy(path, link, nlink);
cleannname(path);

o = crumb(aux, ndotdot)->obj;
assert(o->type == GTree);
for(p = path; *p; p = e){
n = nil;
e = p + strchr(p, "/");

```

if(*e == '/')
    *e++ = '\\0';
/*
 * cleannname guarantees these show up at the start of the name,
 * which allows trimming them from the end of the trail of crumbs
 * instead of needing to keep track of full parentage.
 */
if(strcmp(p, "..") == 0)
    n = crumb(aux, ++ndotdot)->obj;
else if(o->type == GTree)
    for(i = 0; i < o->tree->nent; i++)
        if(strcmp(o->tree->ent[i].name, p) == 0){
            *mode = o->tree->ent[i].mode;
            n = readobject(o->tree->ent[i].h);
            break;
        }
    o = n;
if(o == nil)
    break;
}
free(path);
for(i = 0; o != nil && i < aux->ncrumb; i++)
    if(crumb(aux, i)->obj == o)
        return nil;
return o;
}

```

Uses GTree 33a, crumb() 67d, and readobject() 157b.

23.2 Splitting a large repository: submodules

```

⟨Dirent other fields 237a⟩+≡ (34f) <236a
    bool_byte ismod;

```

```

⟨parsetree() if submodule 237b⟩≡ (47c)
    if(m == 0160000){ /* module */
        t->mode |= DMDIR;
        t->ismod = true;
    }

```

Uses Cloaded.

```

⟨gitmode() if submodule 237c⟩≡ (58a)
    else if(e->ismod)
        return 0160000;

```

```

⟨gtreegen() if submodule 237d⟩≡ (79b)
    if(e->tree->ent[i].ismod)
        o = emptydir();

```

Uses emptydir().

```

⟨objwalk1() when GTree case, if o->tree->ent[i].ismod 237e⟩≡ (79e)
    if(!w && o->tree->ent[i].ismod)
        w = emptydir();

```

Chapter 24

Advanced Commands

24.1 Tagging a commit with a name

`<Gxxx other cases 238a>+≡ (33a) <177b`
`GTag = 4,`

`<objread() switch o->type cases 238b>+≡ (78b) <80b`
`case GTag:`
`readbuf(r, o->data, o->size);`
`break;`

`<objwalk1() switch o->type cases 238c>+≡ (78a) <81b`
`case GTag:`
`e = Eimpl;`
`break;`

24.1.1 Reading a tag

`<readloose() local types other elements 238d>≡ (44)`
`{"tag", GTag},`

`<parseobject() switch o->type cases 238e>+≡ (45d) <48b`
`case GTag: parsetag(o); break;`

`<function parsetag 238f>≡ (277)`
`static void`
`parsetag(Object *)`
`{`
`}`

Uses `npackf 152c`, `packf 152c`, and `searchindex() 238d`.

24.1.2 Writing a tag

24.2 git/hist

24.2.1 Usage

`<hist.rc command line processing 238g>≡ (239a)`
`flagfmt='n:num count'; args='files...'`
`eval '{auxrc/getflags $*} || exec auxrc/usage`
`if(~ $#* 0) exec auxrc/usage`

24.2.2 hist.rc

```
<git9/hist.rc 239a>≡
#!/bin/rc
rfork ne
. /lib/git/common.rc
gitup

<hist.rc command line processing 238g>

fn dodiff {
    while(t='{read}'){
        h=${t(1)}
        o=${gitfs/object/'{git/query $h~}
        c=${gitfs/object/$h
        echo 'Hash:' $h
        echo -n 'Date: '; date '{walk -em $c/msg}
        echo -n 'Author: '; cat $c/author
        echo -n 'Message: '; cat $c/msg
        echo
        for(f in $files){
            curr=${c/tree/$f
            prev=${o/tree/$f
            if(! test -e $curr)
                curr=/dev/null
            if(! test -e $prev)
                prev=/dev/null
            diff -u $prev $curr
        }
        echo --' '
        echo ☺
        echo
    }
}

files='{cleanname -d $gitrel $*}
args=(-s $files)
if(! ~ $#num 0)
    args=(-n $num $args)
git/log $args | dodiff
```

24.3 Reverting a change: git revert

24.3.1 Usage

24.3.2 revert.rc

```
<git9/revert.rc 239b>≡
#!/bin/rc
rfork en
. /lib/git/common.rc

gitup

flagfmt='c:query query' args='file ...'
if (! eval '{auxrc/getflags $*} || ~ $#* 0)
    exec auxrc/usage
```

```

if(~ $#query 0)
    query=HEAD
commit='{git/query -p $query}'

files='${nl}{cleanname -d $gitrel $* | drop $gitroot}'
for(f in '{git/walk -c -fRM -b $query $files}'){
    mkdir -p '{basename -d $f}'
    cp -x -- $commit/tree/$f $f
    touch -c $f
    git/add $f
}
exit ''

```

24.4 Rebase: git rebase

24.4.1 Usage

24.4.2 rebase.rc

```

<git9/rebase.rc 240>≡
#!/bin/rc -e

rfork en

. /lib/git/common.rc
gitup

flagfmt='s:src src, n:nocommit'; args='onto'
eval '{auxrc/getflags $*}' || exec auxrc/usage
if(~ $#* 0) exec auxrc/usage

if(~ $#src 0)
    src='{git/branch}'
nflag=''
if(~ $nocommit 1)
    nflag='-n'

dst='{git/query $1}'
com='{git/query $dst $src @}'
if(~ $dst $com)
    die 'nothing to rebase, doofus'
git/log -se $dst'..' $src | awk '
BEGIN{
    src=ENVIRON["src"];
    dst=ENVIRON["dst"];
}
{
    if(!done)
        print "git/branch -nb "dst" rebase.wip";
    c=$1; $1="";
    print "git/export "c" | git/import '$nflag'#" $0;
    done=1
}
END{
    if(!done)
        print "git/branch -nb "dst" ";
    else{

```

```
    print "git/branch -nb rebase.wip "src
    print "git/branch -r rebase.wip";
  }
}'
```

Chapter 25

Advanced Networking

25.1 Other clients

25.1.1 http://

```
<Connxxx other cases 242a>≡ (181d) 245e▷  
    ConnHttp,
```

```
<Conn http fields 242b>≡ (181c)  
    /* only used by http */  
    fdt cfd;  
    char *url; /* note, first GET uses a different url */  
    char *dir;  
    char *direction;
```

```
<gitconnect() when uri not local repo, if series on proto cases 242c>≡ (206b) 246a▷  
    else if(strcmp(proto, "http") == 0 || strcmp(proto, "https") == 0)  
        return dialhttp(c, host, port, path, direction);
```

```
<parseuri() if series on proto cases 242d>≡ (206c) 242e▷  
    else if(strncmp(proto, "http", 4) == 0)  
        snprintf(port, Nport, "80");
```

Uses Nport-34 279.

```
<parseuri() if series on proto cases 242e>+≡ (206c) <242d 246g▷  
    else if(strncmp(proto, "https", 5) == 0)  
        snprintf(port, Nport, "443");
```

Uses Nport-34 279.

```
<closeconn() switch c->type cases 242f>+≡ (185c) <208c 246c▷  
    case ConnHttp:  
        close(c->cfd);  
        break;
```

dialhttp()

```
<function dialhttp 242g>≡ (279)  
    static int  
    dialhttp(Conn *c, char *host, char *port, char *path, char *direction)  
    {  
        char *geturl, *suff, *hsep, *psep, *isep;  
  
        suff = "";  
        hsep = "";
```

```

psep = "";
isep = "";
if(port && strlen(port) != 0)
    hsep = ":";
if(path && path[0] != '/')
    psep = "/";
if(path && path[0] && path[strlen(path)-1] != '/')
    isep = "/";
memset(c, 0, sizeof(*c));
geturl = smprintf("https://%s%s%s%s%s%s/sinfo/refs?service=git-%s-pack",
    host, hsep, port, psep, path, suff, isep, direction);
c->type = ConnHttp;
c->url = smprintf("https://%s%s%s%s%s%s/sgit-%s-pack",
    host, hsep, port, psep, path, suff, isep, direction);

c->cfid = webclone(c, geturl);

free(geturl);
if(c->cfid == -1)
    return -1;
c->rfd = webopen(c, "body", OREAD);
c->wfd = -1;
if(c->rfd == -1)
    return -1;
if(issmarthttp(c, direction) == -1)
    return -1;
c->direction = estrdup(direction);
return 0;
}

```

Uses ConnHttp, estrdup() 257b, issmarthttp(), webclone() 206c, and webopen() 243b.

```

<function webopen 243a>≡ (279)
static int
webopen(Conn *c, char *file, int mode)
{
    char path[128];
    int fd;

    snprintf(path, sizeof(path), "%s/%s", c->dir, file);
    if((fd = open(path, mode)) == -1)
        return -1;
    return fd;
}

```

webclone()

```

<function webclone 243b>≡ (279)
static int
webclone(Conn *c, char *url)
{
    char buf[16];
    int n, conn;

    if((c->cfid = open("/mnt/web/clone", ORDWR)) < 0)
        goto err;
    if((n = read(c->cfid, buf, sizeof(buf)-1)) == -1)
        goto err;
    buf[n] = 0;
    conn = atoi(buf);
}

```

```

/* github will behave differently based on useragent */
if(write(c->cfid, Useragent, sizeof(Useragent)) == -1)
    return -1;
dprint(1, "open url %s\n", url);
if(fprint(c->cfid, "url %s", url) == -1)
    goto err;
free(c->dir);
c->dir = smprint("/mnt/web/%d", conn);
return 0;
err:
if(c->cfid != -1)
    close(c->cfid);
return -1;
}

```

Uses Useragent-30 [244a](#) and dprint [270](#).

```

⟨constant Useragent 244a⟩≡ (279)
#define Useragent "useragent git/2.24.1"

```

issmarthttp()

```

⟨function issmarthttp 244b⟩≡ (279)

```

```

static int
issmarthttp(Conn *c, char *direction)
{
    char buf[Pktmax+1], svc[128];
    int fd, n;

    if((fd = webopen(c, "contenttype", OREAD)) == -1)
        return -1;
    n = readn(fd, buf, sizeof(buf) - 1);
    close(fd);
    if(n == -1)
        return -1;
    buf[n] = '\0';
    snprintf(svc, sizeof(svc), "application/x-git-%s-pack-advertisement", direction);
    if(strcmp(svc, buf) != 0){
        werrstr("dumb http protocol not supported");
        return -1;
    }

    if((n = readpkt(c, buf, sizeof(buf))) == -1)
        sysfatal("http read: %r");
    buf[n] = 0;
    snprintf(svc, sizeof(svc), "# service=git-%s-pack\n", direction);
    if(strncmp(svc, buf, n) != 0){
        werrstr("invalid initial packet line");
        return -1;
    }
    if(readpkt(c, buf, sizeof(buf)) != 0){
        werrstr("protocol garble: expected flushpkt");
        return -1;
    }
    return 0;
}

```

Uses readpkt() [209c](#) and webopen() [243b](#).

writephase()

<constant Contenthdr 245a>≡ (279)
#define Contenthdr "headers Content-Type: application/x-git-%s-pack-request"

<constant Accepthdr 245b>≡ (279)
#define Accepthdr "headers Accept: application/x-git-%s-pack-result"

<function writephase 245c>≡ (279)
int
writephase(Conn *c)
{
 char hdr[128];
 int n;

 dprint(1, "start write phase\n");
 if(c->type != ConnHttp)
 return 0;

 if(c->wfd != -1)
 close(c->wfd);
 if(c->cfid != -1)
 close(c->cfid);
 if((c->cfid = webclone(c, c->url)) == -1)
 return -1;
 n = snprintf(hdr, sizeof(hdr), Contenthdr, c->direction);
 if(write(c->cfid, hdr, n) == -1)
 return -1;
 n = snprintf(hdr, sizeof(hdr), Accepthdr, c->direction);
 if(write(c->cfid, hdr, n) == -1)
 return -1;
 if((c->wfd = webopen(c, "postbody", OWRITE)) == -1)
 return -1;
 c->rfd = -1;
 return 0;
}

Uses ConnGit 249a, ConnGit9, ConnHttp, ConnSsh 33c, dprint 270, and webopen() 243b.

readphase()

<function readphase 245d>≡ (279)
int
readphase(Conn *c)
{
 dprint(1, "start read phase\n");
 if(c->type != ConnHttp)
 return 0;
 if(close(c->wfd) == -1)
 return -1;
 if((c->rfd = webopen(c, "body", OREAD)) == -1)
 return -1;
 c->wfd = -1;
 return 0;
}

25.1.2 ssh://

<Connxxx other cases 245e>+≡ (181d) <242a 246d>
ConnSsh,

`<gitconnect() when uri not local repo, if series on proto cases 246a>+≡ (206b) <242c 246e>`

```
else if(strcmp(proto, "ssh") == 0)
    return dialssh(c, host, port, path, direction);
```

Uses Contenthdr-31 245a.

`<function dialssh 246b>≡ (279)`

```
static int
dialssh(Conn *c, char *host, char *, char *path, char *direction)
{
    int pid, pfd[2];
    char cmd[64];

    if(pipe(pfd) == -1)
        sysfatal("unable to open pipe: %r");
    pid = fork();
    if(pid == -1)
        sysfatal("unable to fork");
    if(pid == 0){
        close(pfd[1]);
        dup(pfd[0], 0);
        dup(pfd[0], 1);
        snprintf(cmd, sizeof(cmd), "git-%s-pack", direction);
        dprint(1, "exec ssh '%s' '%s' %s\n", host, cmd, path);

        execl("/bin/ssh", "ssh", host, cmd, path, nil);
        sysfatal("exec: %r");
    }
    close(pfd[0]);
    c->type = ConnSsh;
    c->rfd = pfd[1];
    c->wfd = dup(pfd[1], -1);
    return 0;
}
```

Uses ConnSsh 33c and dprint 270.

`<closeconn() switch c->type cases 246c>+≡ (185c) <242f`

```
case ConnGit9:
case ConnSsh:
    free(wait());
    break;
```

25.1.3 gits://

`<Connxxx other cases 246d>+≡ (181d) <245e`

```
ConnGit9,
```

`<gitconnect() when uri not local repo, if series on proto cases 246e>+≡ (206b) <246a 246f>`

```
else if(strcmp(proto, "hgit") == 0)
    return dialhgit(c, host, port, path, direction, 1);
```

Uses Accepthdr-32 245b.

`<gitconnect() when uri not local repo, if series on proto cases 246f>+≡ (206b) <246e`

```
else if(strcmp(proto, "gits") == 0)
    return dialhgit(c, host, port, path, direction, 0);
```

Uses webopen() 243b.

`<parseuri() if series on proto cases 246g>+≡ (206c) <242e 247a>`

```
else if(strncmp(proto, "gits", 5) == 0)
    snprintf(port, Nport, "9419");
```

Uses Nport-34 279.

`<parseuri() if series on proto cases 247a>+≡ (206c) <246g`

```
else if(strncmp(proto, "hgit", 5) == 0)
    snprintf(port, Nport, "17021");
```

`<function dialhgit 247b>≡ (279)`

```
static int
dialhgit(Conn *c, char *host, char *port, char *path, char *direction, int auth)
{
    char *ds;
    int pid, pfd[2];

    if((ds = netmkaddr(host, "tcp", port)) == nil)
        return -1;
    if(pipe(pfd) == -1)
        sysfatal("unable to open pipe: %r");
    pid = fork();
    if(pid == -1)
        sysfatal("unable to fork");
    if(pid == 0){
        close(pfd[1]);
        dup(pfd[0], 0);
        dup(pfd[0], 1);
        dprint(1, "exec tlsclient -a %s\n", ds);
        if(auth)
            execl("/bin/tlsclient", "tlsclient", "-a", ds, nil);
        else
            execl("/bin/tlsclient", "tlsclient", ds, nil);
        sysfatal("exec: %r");
    }
    close(pfd[0]);
    c->type = ConnGit9;
    c->rfd = pfd[1];
    c->wfd = dup(pfd[1], -1);
    return githandshake(c, host, path, direction);
}
```

Uses `ConnGit 249a`, `ConnGit9`, `dprint 270`, and `githandshake() 246b`.

Chapter 26

Advanced Topics

26.1 Bytes versus ASCII versus UTF-8

26.2 Reliability and signals

26.3 Optimizations

26.3.1 Object cache

`<global objcache 248a>≡ (277)`
`Objset objcache;`

`<gitinit() set objcache 248b>≡ (61f)`
`osinit(&objcache);`

Uses `objcache 152a` and `osinit() 37c`.

`<Cxxx other cases 248c>+≡ (33c) <180c 249a>`
`Ccache = 1 << 4,`

`<clear() assert o->flag 248d>+≡ (34a) <34b`
`assert((o->flag & Ccache) == 0);`

`<Object cache fields 248e>+≡ (32b) <33b`
`int id;`

`<readidxobject() if h in object cache 248f>≡ (43b)`
`if((obj = osfind(&objcache, h)) != nil){`
`if(flag & Cidx){`
`/*`
`* If we're indexing, we need to be careful`
`* to only return objects within this pack,`
`* so if the objects exist outside the pack,`
`* we don't index the wrong copy.`
`*/`
`if(!(obj->flag & Cidx))`
`return nil;`
`if(obj->flag & Cloaded)`
`return obj;`
`o = Boffset(idx);`
`if(Bseek(idx, obj->off, 0) == -1)`
`return nil;`
`if(readpacked(idx, obj, flag) == -1)`
`return nil;`

```

    if(Bseek(idx, o, 0) == -1)
        sysfatal("could not restore offset");
    cache(obj);
    return obj;
}
if(obj->flag & Cloaded)
    return obj;
}

```

Uses `npackf` 152c, `packf` 152c, `refreshpacks()`, and `searchindex()` 238d.

<Cxxx other cases 249a> ≡ (33c) <248c
`Cexist = 1 << 5,`

<globals lruxxx 249b> ≡ (277)
`Object *lruhead;`
`Object *lrutail;`
`vlong ncache;`

<Object extra fields 249c> ≡ (32b)
`Object *next;`
`Object *prev;`

<function cache 249d> ≡ (277)
`void`
`cache(Object *o)`
`{`
 `Object *p;`

 `if(o == lruhead)`
 `return;`
 `if(o == lrutail)`
 `lrutail = lrutail->prev;`
 `if(!(o->flag & Cexist)){`
 `osadd(&objcache, o);`
 `o->id = objcache.nobj;`
 `o->flag |= Cexist;`
 `}`
 `if(o->prev != nil)`
 `o->prev->next = o->next;`
 `if(o->next != nil)`
 `o->next->prev = o->prev;`
 `if(lrutail == o){`
 `lrutail = o->prev;`
 `if(lrutail != nil)`
 `lrutail->next = nil;`
 `}else if(lrutail == nil)`
 `lrutail = o;`
 `if(lruhead)`
 `lruhead->prev = o;`
 `o->next = lruhead;`
 `o->prev = nil;`
 `lruhead = o;`

 `if(!(o->flag & Ccache)){`
 `o->flag |= Ccache;`
 `ref(o);`
 `ncache += o->size;`
 `}`
 `while(ncache > cachemax && lrutail != lruhead){`
 `p = lrutail;`

```

    lrutail = p->prev;
    if(lrutail != nil)
        lrutail->next = nil;
    p->flag &= ~Ccache;
    p->prev = nil;
    p->next = nil;
    ncache -= p->size;
    unref(p);
}
}

```

Uses Ccache 33c, cachemax, lruhead 152a, lrutail 152a, ncache 152a, npackf 152c, packf 152c, and unref() 34a.

<function clearedobject 250> ≡ (277)

```

/*
 * Creates and returns a cached, cleared object
 * that will get loaded some other time. Useful
 * for performance if need to mark that a blob
 * exists, but we don't care about its contents.
 *
 * The refcount of the returned object is 0, so
 * it doesn't need to be unrefed.
 */
Object*
clearedobject(Hash h, int type)
{
    Object *o;

    if((o = osfind(&objcache, h)) != nil)
        return o;

    o = emalloc(sizeof(Object));
    o->hash = h;
    o->type = type;
    osadd(&objcache, o);
    o->id = objcache.nobj;
    o->flag |= Cexist;
    return o;
}

```

Uses GETBE32, emalloc(), interactive, and showprogress() 39d.

26.4 Fast import and export

26.5 Alternates

26.6 Other repository format

26.6.1 Bare repository

26.6.2 .git file

26.7 Security

Chapter 27

Conclusion

Appendix A

Debugging

A.1 Format dumpers

```
<pragmas varargck 252a>+≡ (270) <31a
#pragma varargck type "T" int
#pragma varargck type "O" Object*
#pragma varargck type "Q" Qid
```

```
<gitinit() fmtinstall calls 252b>+≡ (61f) <31b
fmtinstall('T', Tfmt);
fmtinstall('O', Ofmt);
fmtinstall('Q', Qfmt);
```

Uses Ofmt() 252d, Qfmt() 253a, and Tfmt() 252c.

```
<function Tfmt 252c>≡ (284b)
int
Tfmt(Fmt *fmt)
{
    int t;
    int l;

    t = va_arg(fmt->args, int);
    switch(t){
    case GNone: l = fmtprint(fmt, "none"); break;
    case GCommit: l = fmtprint(fmt, "commit"); break;
    case GTree: l = fmtprint(fmt, "tree"); break;
    case GBlob: l = fmtprint(fmt, "blob"); break;
    case GTag: l = fmtprint(fmt, "tag"); break;
    case GDelta: l = fmtprint(fmt, "delta"); break;
    case GRdelta: l = fmtprint(fmt, "gdelta"); break;
    default: l = fmtprint(fmt, "?%d?", t); break;
    }
    return l;
}
```

Uses GBlob 33a, GCommit 33a, GNone 270, GDelta 33a, GRdelta 33a, GTag 177b, and GTree 33a.

```
<function Ofmt 252d>≡ (284b)
int
Ofmt(Fmt *fmt)
{
    Object *o;
    int l;

    o = va_arg(fmt->args, Object *);
    print("== %H (%T) ==\n", o->hash, o->type);
```

```

switch(o->type){
case GTree:
    l = fmtprint(fmt, "tree\n");
    break;
case GBlob:
    l = fmtprint(fmt, "blob %s\n", o->data);
    break;
case GCommit:
    l = fmtprint(fmt, "commit\n");
    break;
case GTag:
    l = fmtprint(fmt, "tag\n");
    break;
default:
    l = fmtprint(fmt, "invalid: %d\n", o->type);
    break;
}
return l;
}

```

Uses GBlob 33a, GCommit 33a, GTag 177b, and GTree 33a.

```

⟨function Qfmt 253a⟩≡ (284b)
int
Qfmt(Fmt *fmt)
{
    Qid q;

    q = va_arg(fmt->args, Qid);
    if(q.path == ~OULL && q.vers == ~OUL && q.type == 0xff)
        return fmtprint(fmt, "NOQID");
    else
        return fmtprint(fmt, "%llx.%ld.%hxx", q.path, q.vers, q.type);
}

```

A.2 git/fs -d

```

⟨main() (fs.c) command line processing 253b⟩+≡ (65b) <86b
    case 'd':
        chatty9p++;
        break;

```

A.3 git/xxx -d

```

⟨global chattygit 253c⟩≡ (284b)
int chattygit;

```

```

⟨macro dprint 253d⟩≡ (270)
#define dprint(lvl, ...) \
    if(chattygit >= lvl) _dprint(__VA_ARGS__)

```

```

⟨function _dprint 253e⟩≡ (284b)
void
_dprint(char *fmt, ...)
{
    va_list ap;

```

```

    va_start(ap, fmt);
    vfprintf(STDERR, fmt, ap);
    va_end(ap);
}

```

`<main() (query.c) command line processing 254a>+≡ (87b) <101g`

```

    case 'd':  chattygit++;  break;

```

`<main() (get.c) command line processing 254b>+≡ (184d) <196d`

```

    case 'd':  chattygit++;  break;

```

`<main() (repack.c) command line processing 254c>≡ (149b)`

```

    case 'd':  chattygit++;  break;

```

`<main() (send.c) command line processing 254d>+≡ (198b) <203c`

```

    case 'd':  chattygit++;  break;

```

`<main() (serve.c) command line processing 254e>+≡ (211c) <222b`

```

    case 'd':  chattygit++;  break;

```

A.4 Tracing packets

`<function tracepkt 254f>≡ (279)`

```

void
tracepkt(int v, char *pfx, char *b, int n)
{
    char *f;
    int o, i;

    if(chattygit < v)
        return;
    o = 0;
    f = emalloc(n*4 + 1);
    for(i = 0; i < n; i++){
        if(isprint(b[i])){
            f[o++] = b[i];
            continue;
        }
        f[o++] = '\\';
        switch(b[i]){
            case '\\': f[o++] = '\\'; break;
            case '\n': f[o++] = 'n'; break;
            case '\r': f[o++] = 'r'; break;
            case '\v': f[o++] = 'v'; break;
            case '\0': f[o++] = '0'; break;
            default:
                f[o++] = 'x';
                f[o++] = "0123456789abcdef"[(b[i]>>4)&0xf];
                f[o++] = "0123456789abcdef"[(b[i]>>0)&0xf];
                break;
        }
    }
    f[o] = '\0';
    fprintf(STDERR, "%s %04x:\t%s\n", pfx, n, f);
    free(f);
}

```

Uses `chattygit 253c`.

Appendix B

Error Management

```
<fn die(common.rc 255a)≡ (261a)
  fn die{
    >[1=2] echo $0: $*
    exit $"*
  }
```

```
<constant Egreg 255b)≡ (267)
  #define Egreg "wat"
```

```
<constant Enodir 255c)≡ (267)
  #define Enodir "not a directory"
```

```
<constant Eperm 255d)≡ (267)
  #define Eperm "permission denied"
```

```
<constant Eexist 255e)≡ (267)
  #define Eexist "file does not exist"
```

```
<constant Enotdir 255f)≡ (267)
  #define Enotdir "is not a directory"
```

```
<constant E2long 255g)≡ (267)
  #define E2long "path too long"
```

```
<constant Erepo 255h)≡ (267)
  #define Erepo "unable to read repo"
```

```
<constant Eimpl 255i)≡ (267)
  #define Eimpl "not implemented"
```

```
<constant Ebadobj 255j)≡ (267)
  #define Ebadobj "invalid object"
```

Appendix C

Utilities

<macro `max` [256a](#))≡ (277)
#define max(x, y) ((x) > (y) ? (x) : (y))

C.1 Error malloc

<function `emalloc` [256b](#))≡ (284b)
void *
emalloc(ulong n)
{
 void *v;

 v = mallocz(n, 1);
 if(v == nil)
 sysfatal("malloc: %r");
 setmalloctag(v, getcallerpc(&n));
 return v;
}

<function `eamalloc` [256c](#))≡ (284b)
void *
eamalloc(ulong n, ulong sz)
{
 uulong na;
 void *v;

 na = (uulong)n*(uulong)sz;
 if(na >= (1ULL<<30))
 sysfatal("alloc: overflow");
 v = mallocz(na, 1);
 if(v == nil)
 sysfatal("malloc: %r");
 setmalloctag(v, getcallerpc(&n));
 return v;
}

<function `erealloc` [256d](#))≡ (284b)
void *
erealloc(void *p, ulong n)
{
 void *v;

 v = realloc(p, n);
 if(v == nil)

```

        sysfatal("realloc: %r");
    setmalloctag(v, getcallerpc(&p));
    return v;
}

```

<function earealloc 257a>≡ (284b)

```

void *
earealloc(void *p, ulong n, ulong sz)
{
    uulong na;
    void *v;

    na = (uulong)n*(uulong)sz;
    if(na >= (1ULL<<30))
        sysfatal("alloc: overflow");
    v = realloc(p, na);
    if(v == nil)
        sysfatal("realloc: %r");
    setmalloctag(v, getcallerpc(&p));
    return v;
}

```

<function estrdup 257b>≡ (284b)

```

char*
estrdup(char *s)
{
    s = strdup(s);
    if(s == nil)
        sysfatal("strdup: %r");
    setmalloctag(s, getcallerpc(&s));
    return s;
}

```

<function emalloc(diff) 257c>≡ (311b)

```

void *
emalloc(unsigned n)
{
    register void *p;

    if ((p = malloc(n)) == 0)
        sysfatal("malloc: %r");
    return p;
}

```

<function erealloc(diff) 257d>≡ (311b)

```

void *
erealloc(void *p, unsigned n)
{
    void *rp;

    if ((rp = realloc(p, n)) == 0)
        sysfatal("realloc: %r");
    return rp;
}

```

<function emalloc(patch.c) 257e>≡ (323b)

```

void *
emalloc(ulong n)
{
    void *v;

```

```

    v = mallocz(n, 1);
    if(v == nil)
        fail("malloc: %r");
    setmalloctag(v, getcallerpc(&n));
    return v;
}

```

<function erealloc(patch.c) 258a>≡ (323b)

```

void *
erealloc(void *v, ulong n)
{
    if(n == 0)
        n++;
    v = realloc(v, n);
    if(v == nil)
        fail("malloc: %r");
    setmalloctag(v, getcallerpc(&n));
    return v;
}

```

C.2 String utilities

<macro isblank 258b>≡ (270)

```

#define isblank(c) \
    (((c) != '\n') && isspace(c))

```

<function strip 258c>≡ (284b)

```

char *
strip(char *s)
{
    char *e;

    while(isspace(*s))
        s++;
    e = s + strlen(s);
    while(e > s && isspace(*--e))
        *e = 0;
    return s;
}

```

<function nextline (git9/pack.c) 258d>≡ (277)

```

static void
nextline(char **str, int *nstr)
{
    char *s;

    if((s = strchr(*str, '\n')) != nil){
        *nstr -= s - *str + 1;
        *str = s + 1;
    }
}

```

Uses `hparse()` 31d and `parseauthor()` 155c.

```

⟨function nextline 259a⟩≡ (272)
static char*
nextline(char *p, char *e)
{
    for(; p != e; p++)
        if(*p == '\n')
            break;
    return p;
}

```

Uses out 143a.

```

⟨function scanword 259b⟩≡ (277)
/*
 * Scans for non-empty word, copying it into buf.
 * Strips off word, leading, and trailing space
 * from input.
 *
 * Returns -1 on empty string or error, leaving
 * input unmodified.
 */
static errorneg1
scanword(char **str, int *nstr, char *buf, int nbuf)
{
    char *p;
    int n;
    errorneg1 r;

    r = ERROR_NEG1;
    p = *str;
    n = *nstr;
    while(n && isblank(*p)){
        n--;
        p++;
    }

    for(; n && *p && !isspace(*p); p++, n--){
        r = OK_0;
        *buf++ = *p;
        nbuf--;
        if(nbuf == 0)
            return ERROR_NEG1;
    }
    while(n && isblank(*p)){
        n--;
        p++;
    }
    *buf = '\0';
    *str = p;
    *nstr = n;
    return r;
}

```

Uses authorpat 284b and hparse() 31d.

```

⟨function endswith 259c⟩≡ (276a)
bool
endswith(char *n, char *s)
{
    int nn, ns;

    nn = strlen(n);

```

```
    ns = strlen(s);
    return nn > ns && strcmp(n + nn - ns, s) == 0;
}
```

C.3 File and directory utilities

```
<function slurpdir 260>≡ (284b)
int
slurpdir(char *p, Dir **d)
{
    int r;
    fdt f;

    f = open(p, OREAD);
    if(f == ERROR_NEG1)
        return ERROR_NEG1;
    r = dirreadall(f, d);
    close(f);
    return r;
}
```

Appendix D

Extra Code

D.1 git9/

D.1.1 git9/common.rc

```
<git9/common.rc 261a>≡  
  <global nl(common.rc) 126a>  
  
  <fn die(common.rc) 255a>  
  
  <fn subst(common.rc) 130b>  
  <fn drop(common.rc) 126c>  
  
  <fn whoami(common.rc) 128a>  
  
  <fn mergeperm(common.rc) 138c>  
  <fn merge1(common.rc) 138b>  
  
  <fn gitup(common.rc) 125d>
```

D.1.2 git9/add.rc

D.1.3 git9/branch.rc

D.1.4 git9/clone.rc

D.1.5 git9/commit.rc

D.1.6 git9/compat.rc

```
<git9/compat.rc 261b>≡  
#!/bin/rc  
  
rfork e  
  
opts=()  
args=()  
nl=''  
,  
  
fn cmd_init{  
  while(! ~ $#* 0){  
    switch($1){
```

```

        case --bare
            # ignore
        case --
            # go likes to use these
        case -*
            die unknown command init $*
        case *
            args=($args $1)
    }
    shift
}
git/init $opts $args
}

fn cmd_clone{
    branch=()
    while( ! ~ $#* 0){
        switch($1){
            case -b
                branch=$2
                shift
            case --
                # go likes to use these
            case -*
                die unknown command clone $*
            case *
                args=($args $1)
        }
        shift
    }
    git/clone $opts $args
    if(~ $#branch 1)
        git/branch -n -b $1 origin/$1
}

fn cmd_pull{
    if(~ $1 -*)
        die unknown options for pull $*
    git/pull
}

fn cmd_fetch{
    while(~ $#* 0){
        switch($1){
            case --all
                opts=($opts -a)
            case -f
                opts=($opts -u $2)
                shift
            case --
                # go likes to use these
            case -*
                die unknown command clone $*
            case *
                args=($args $1)
        }
        shift
    }
    git/pull -f $opts
}

```

```

fn cmd_checkout{
    if(~ $1 -*)
        die unknown command pull $*
    if(~ $#* 0)
        die git checkout branch
    git/branch $b
}

fn cmd_submodule {
    if(test -f .gitmodules)
        die 'submodules unsupported'
}

fn cmd_rev-parse{
    while(~ $1 -*){
        switch($1){
            case --git-dir
                echo $gitroot/.git
                shift
            case --abbrev-ref
                echo '{dcmd git9/branch | sed s@^heads/@@g}'
                shift
            case *
                die unknown option $opt
        }
        shift
    }
}

fn cmd_show-ref{
    if(~ $1 -*)
        die unknown command pull $*
    filter=cat
    if(~ $#* 0)
        filter=cat
    if not
        filter='-e(^|/)'~$*~'$'
    for(b in '$nl{cd $gitroot/.git/refs/ && walk -f}')
        echo '{cat $gitroot/.git/refs/$b} refs/$b'
}

fn cmd_rev-parse{
    switch($1){
        case --git-dir
            echo '{git/conf -r}^/.git'
        case *
            die 'unknown rev-parse '$*
    }
}

fn cmd_remote{
    if({! ~ $#* 3 && ! ~ $#* 4} || ! ~ $1 add)
        die unimplemented remote cmd $*
    name=$2
    url=$3
    if(~ $3 '--')
        url=$4
    >>$gitroot/.git/config{

```

```

        echo '[remote "'$name']'
        echo ' url='$url
    }
}

fn cmd_log{
    count=()
    format=''
    while(~ $1 -*){
        switch($1){
            case --format
                format=$2
                shift
            case '--format=*'
                format='{echo $1 | sed 's/--format=//g'}'
            case -n
                count=-n$2
                shift
            case -n*
                count=$1
            case *
                dprint option $opt
        }
        shift
    }
    @{cd $gitroot && git/fs}
    switch($format){
        case ''
            git/log $count
        case '%H:%ct'
            for(c in '{git/log -s $count | awk '{print $1}'}')
                echo $c: '{mtime $gitroot/.git/fs/object/$c/msg}'
        case '%h %cd'
            for(c in '{git/log -s $count | awk '{print $1}'}')
                echo $c '{date '{mtime $gitroot/.git/fs/object/$c/msg}}'
    }
}

}

fn cmd_show{
    cmd_log -n1 $*
}

}

fn cmd_ls-remote{
    if(~ $1 -q)
        shift
    remote='${nl}{git/conf 'remote "'$1".url'}'
    if(~ $#remote 0)
        remote=$1
    git/get -l $remote | awk '/^remote/{print $3"\t"$2}'
}

}

fn cmd_version{
    echo git version 2.2.0
}

}

fn cmd_status{
    echo
}

}

```

```

fn usage{
    echo 'git <command> <args>' >[1=2]
    exit usage
}

fn die {
    >[1=2] echo git $_cmdname: $*
    exit $_cmdname: $*
}

_cmdname=$1
if(~ $0 *compat){
    ramfs -m /n/gitcompat
    touch /n/gitcompat/git
    bind $0 /n/gitcompat/git
    path=( /n/gitcompat $path )
    exec rc
}

if(~ $#gitcompatdebug 1)
    echo running $* >>/tmp/gitlog

if(~ $1 -c)
    shift 2
if(~ $1 -c*)
    shift 1
if(! test -f '/env/fn#cmd_'$1)
    die git $1: commmand not implemented
if(! ~ $1 init && ! ~ $1 clone)
    gitroot='{git/conf -r} || die repo

if(~ $#gitcompatdebug 1)
    cmd_$1 $*(2-) | tee >>/tmp/gitlog
if not
    cmd_$1 $*(2-)
exit ''

```

D.1.7 git9/diff.rc

D.1.8 git9/export.rc

D.1.9 git9/import.rc

D.1.10 git9/hist.rc

D.1.11 git9/init.rc

D.1.12 git9/merge.rc

D.1.13 git9/pull.rc

D.1.14 git9/push.rc

D.1.15 git9/rebase.rc

D.1.16 git9/revert.rc

D.1.17 git9/rm.rc

D.1.18 git9/conf.c

<git9/conf.c 266a>≡

```
#include <u.h>
#include <libc.h>
#include <ctype.h>
```

```
#include "git.h"
```

```
<global findroot 62b>
```

```
<global showall 63b>
```

```
<global nfile 61c>
```

```
<global file 61b>
```

```
<function showconf 63a>
```

```
<function usage 60a>
```

```
<function main 60b>
```

D.1.19 git9/delta.c

<git9/delta.c 266b>≡

```
#include <u.h>
#include <libc.h>
```

```
#include "git.h"
```

```
enum {
    Minchunk    = 128,
    Maxchunk    = 8192,
    Splitmask   = (1<<8)-1,
};
```

```
static u32int geartab[] = {
    0x67ed26b7, 0x32da500c, 0x53d0fee0, 0xce387dc7, 0xcd406d90, 0x2e83a4d4, 0x9fc9a38d, 0xb67259dc,
```

```

0xca6b1722, 0x6d2ea08c, 0x235cea2e, 0x3149bb5f, 0x1beda787, 0x2a6b77d5, 0x2f22d9ac, 0x91fc0544,
0xe413acfa, 0x5a30ff7a, 0xad6fdde0, 0x444fd0f5, 0x7ad87864, 0x58c5ff05, 0x8d2ec336, 0x2371f853,
0x550f8572, 0x6aa448dd, 0x7c9ddbcf, 0x95221e14, 0x2a82ec33, 0xcbec5a78, 0xc6795a0d, 0x243995b7,
0x1c909a2f, 0x4fded51c, 0x635d334b, 0x0e2b9999, 0x2702968d, 0x856de1d5, 0x3325d60e, 0xeb6a7502,
0xec2a9844, 0x0905835a, 0xa1820375, 0xa4be5cab, 0x96a6c058, 0x2c2ccd70, 0xba40fce3, 0xd794c46b,
0x8fbae83e, 0xc3aa7899, 0x3d3ff8ed, 0xa0d42b5b, 0x571c0c97, 0xd2811516, 0xf7e7b96c, 0x4fd2fcdb,
0xe2fdec94, 0x282cc436, 0x78e8e95c, 0x80a3b613, 0xcfbee20c, 0xd4a32d1c, 0x2a12ff13, 0x6af82936,
0xe5630258, 0x8efa6a98, 0x294fb2d1, 0xdeb57086, 0x5f0fddb3, 0xeceda7ce, 0x4c87305f, 0x3a6d3307,
0xe22d2942, 0x9d060217, 0x1e42ed02, 0xb6f63b52, 0x4367f39f, 0x055cf262, 0x03a461b2, 0x5ef9e382,
0x386bc03a, 0x2a1e79c7, 0xf1a0058b, 0xd4d2dea9, 0x56baf37d, 0x5daff6cc, 0xf03a951d, 0xaef7de45,
0xa8f4581e, 0x3960b555, 0xffbfff6d, 0xbe702a23, 0x8f5b6d6f, 0x061739fb, 0x98696f47, 0x3fd596d4,
0x151eac6b, 0xa9fcc4f5, 0x69181a12, 0x3ac5a107, 0xb5198fe7, 0x96bcb1da, 0x1b5ddf8e, 0xc757d650,
0x65865c3a, 0x8fc0a41a, 0x87435536, 0x99eda6f2, 0x41874794, 0x29cff4e8, 0xb70efd9a, 0x3103f6e7,
0x84d2453b, 0x15a450bd, 0x74f49af1, 0x60f664b1, 0xa1c86935, 0xfdafbce1, 0xe36353e3, 0x5d9ba739,
0xbc0559ba, 0x708b0054, 0xd41d808c, 0xb2f31723, 0x9027c41f, 0xf136d165, 0xb5374b12, 0x9420a6ac,
0x273958b6, 0xe6c2fad0, 0xebdc1f21, 0xfb33af8b, 0xc71c25cd, 0xe9a2d8e5, 0xbeb38a50, 0xbceb7cc2,
0x4e4e73f0, 0xcd6c251d, 0xde4c032c, 0x4b04ac30, 0x725b8b21, 0x4eb8c33b, 0x20d07b75, 0x0567aa63,
0xb56b2bb7, 0xc1f5fd3a, 0xcafd35ca, 0x470dd4da, 0xfe4f94cd, 0xfb8de424, 0xe8dbc4f0, 0xfe50a37a,
0x62db5b5d, 0xf32f4ab6, 0x2c4a8a51, 0x18473dc0, 0xfe0cbb6e, 0xfe399efd, 0xdf34ecc9, 0x6ccd5055,
0x46097073, 0x139135c2, 0x721c76f6, 0x1c6a94b4, 0x6eee014d, 0x8a508e02, 0x3da538f5, 0x280d394f,
0x5248a0c4, 0x3ce94c6c, 0x9a71ad3a, 0x8493dd05, 0xe43f0ab6, 0x18e4ed42, 0x6c5c0e09, 0x42b06ec9,
0x8d330343, 0xa45b6f59, 0x2a573c0c, 0xd7fd3de6, 0xeedeab68, 0x5c84dafc, 0xbbd1b1a8, 0xa3ce1ad1,
0x85b70bed, 0xb6add07f, 0xa531309c, 0x8f8ab852, 0x564de332, 0xeac9ed0c, 0x73da402c, 0x3ec52761,
0x43af2f4d, 0xd6ff45c8, 0x4c367462, 0xd553bd6a, 0x44724855, 0x3b2aa728, 0x56e5eb65, 0xeaf16173,
0x33fa42ff, 0xd714bb5d, 0xfbd0a3b9, 0xaf517134, 0x9416c8cd, 0x534cf94f, 0x548947c2, 0x34193569,
0x32f4389a, 0xfe7028bc, 0xed73b1ed, 0x9db95770, 0x468e3922, 0x0440c3cd, 0x60059a62, 0x33504562,
0x2b229fbd, 0x5174dca5, 0xf7028752, 0xd63c6aa8, 0x31276f38, 0x0646721c, 0xb0191da8, 0xe00e6de0,
0x9eac1a6e, 0x9f7628a5, 0xed6c06ea, 0x0bb8af15, 0xf119fb12, 0x38693c1c, 0x732bc0fe, 0x84953275,
0xb82ec888, 0x33a4f1b3, 0x3099835e, 0x028a8782, 0x5fdd51d7, 0xc6c717b3, 0xb06caf71, 0x17c8c111,
0x61bad754, 0x9fd03061, 0xe09df1af, 0x3bc9eb73, 0x85878413, 0x9889aaf2, 0x3f5a9e46, 0x42c9f01f,
0x9984a4f4, 0xd5de43cc, 0xd294daed, 0xbecba2d2, 0xf1f6e72c, 0x5551128a, 0x83af87e2, 0x6f0342ba,
};

```

<function hash 168b>

<function addblk 155b>

<function lookup 168a>

<function nextblk 155a>

<function dtinit 154e>

<function dtclear 154d>

<function emitdelta 169a>

<function stretch 168c>

<function deltify 167b>

D.1.20 git9/fs.c

```

<git9/fs.c 267>≡
#include <u.h>
#include <libc.h>
#include <ctype.h>

#include <fcall.h>
#include <thread.h>

```

```

#include <9p.h>

#include "git.h"

<enum Qxxx 66g>

typedef struct Gitaux Gitaux;
typedef struct Crumb Crumb;
typedef struct Cache Cache;
typedef struct Uqid Uqid;

<struct Crumb 67c>
<struct Gitaux 67b>

<struct Uqid 68c>
<struct Cache 69a>

<constant qroot 67a>

<constant Eperm 255d>
<constant Eexist 255e>
<constant Enotdir 255f>
<constant E2long 255g>
<constant Enodir 255c>
<constant Erepo 255h>
<constant Eimpl 255i>
<constant Egreg 255b>
<constant Ebadobj 255j>

<global gitdir 69f>
<global username 66a>
<global groupname 66b>
<global mntpt 66c>
<global branches 66d>
<global uqidcache 69b>
<global nextqid 69d>

static Object* walklink(Gitaux *, char *, int, int, int*);

<function qpath 69c>

<function crumb 67d>

<function popcrumb 74b>

<function branchid 85d>

<function obj2dir 72b>

<function rootgen 74d>

<function branchgen 85e>
<function gtreegen 79b>
<function gcommitgen 80c>

<function objgen 76d>
<function objread 78b>

<function readcommitparent 82c>

```

<function gitattach 70a>
<function walklink 236f>
<function objwalk1 78a>
<function readref 84a>
<function gitwalk1 73>
<function gitclone 70c>
<function gitdestroyfid 71a>
<function readctl 75c>
<function gitread 72c>
<function gitopen 71b>
<function gitstat 72a>

<global gitsrv 66f>

<function usage (git9/fs.c) 65a>
<function main (git9/fs.c) 65b>

Uses Cache, Crumb 67c, Gitaux 67b, and Uqid.

D.1.21 git9/get.c

```
<git9/get.c 269>≡  
#include <u.h>  
#include <libc.h>  
  
#include "git.h"  
  
<global upstream(get.c) 184b>  
<globals heads(get.c) 184c>  
<global listonly(get.c) 196a>  
<global fetchbranch(get.c) 196c>  
  
<function okrefname 190>  
  
<function resolveremote 192a>  
  
<function rename 192b>  
  
<function checkhash 193>  
  
<function mkoutpath 194a>  
  
<function prefixed 194b>  
  
<function branchmatch 194c>  
  
<function fail 194d>  
  
<function enqueueparent 195a>  
  
<function fmtcaps 195b>
```

<function sbread 195c>

<function fetchpack 186c>

<function usage (git9/get.c) 184a>

<function main (git9/get.c) 184d>

Uses `unref()` [34a](#).

D.1.22 git9/git.h

<git9/git.h 270>≡

```
#include <bio.h>
```

```
#include <mp.h>
```

```
#include <libsec.h>
```

```
#include <flate.h>
```

```
#include <regexp.h>
```

```
typedef struct Conn Conn;
```

```
typedef struct Hash Hash;
```

```
typedef struct Delta Delta;
```

```
typedef struct Cinfo Cinfo;
```

```
typedef struct Tinfo Tinfo;
```

```
typedef struct Object Object;
```

```
typedef struct Objset Objset;
```

```
typedef struct Pack Pack;
```

```
typedef struct Buf Buf;
```

```
typedef struct Dirent Dirent;
```

```
typedef struct Idxent Idxent;
```

```
typedef struct Objlist Objlist;
```

```
typedef struct Dtab Dtab;
```

```
typedef struct Dblock Dblock;
```

```
typedef struct Objq Objq;
```

```
typedef struct Qelt Qelt;
```

```
typedef struct Idxent Idxent;
```

```
enum {
```

```
    Pathmax    = 512,
```

```
    <constant Npackcache 161c>
```

```
    Hashsz     = 20,
```

```
    Pktmax     = 65536,
```

```
    KiB       = 1024,
```

```
    MiB       = 1024*KiB,
```

```
};
```

<enum Gxxx 33a>

<enum Cxxx 33c>

<enum Connxxx 181d>

<struct Objlist 68a>

<struct Hash 30a>

<struct Conn 181c>

<struct Dirent 34f>

<struct Object 32b>

```

<struct Tinfo 34e>
<struct Cinfo 36b>

<struct Objset 37b>

<struct Qelt 39a>
<struct Objq 38e>

<struct Dtab 154b>
<struct Dblock 154c>
<struct Delta 153e>

<struct Idxent 41a>

<macro GETBE32 32a>
<macro GETBE64 157a>

<macro PUTBE32 170a>
<macro PUTBE64 176a>

<macro QDIR 69e>
<macro isblank 258b>

extern Reprog    *authorpat;
extern Objset    objcache;
extern vlong     cachemax;
extern Hash      Zhash;
extern int       chattygit;
extern int       interactive;
extern int       gitdirmode;

<pragmas varargck 31a>
int Hfmt(Fmt*);
int Tfmt(Fmt*);
int Ofmt(Fmt*);
int Qfmt(Fmt*);

<signature gitinit 61a>

/* object io */
int resolverefs(Hash **, char *);
int resolveref(Hash *, char *);
int listrefs(Hash **, char ***);
Object *ancestor(Object *, Object *);
int findtwixt(Hash *, int, Hash *, int, Object ***, int *);
Object *readobject(Hash);
Object *clearedobject(Hash, int);
int expandprefix(Hash*, Hash, int);
void parseobject(Object *);
int indexpack(char *, char *, Hash);
int writepack(int, Hash*, int, Hash*, int, Hash*);
int hasheq(Hash *, Hash *);
Object *ref(Object *);
void unref(Object *);
void cache(Object *);
Object *emptydir(void);
int entcmp(void*, void*);

/* object sets */

```

```

void    osinit(Objset *);
void    osclear(Objset *);
void    osadd(Objset *, Object *);
int    oshas(Objset *, Hash);
Object *osfind(Objset *, Hash);

/* object listing */
Objlist *mkols(void);
int    olsnext(Objlist *, Hash *);
void    olsfree(Objlist *);

/* util functions */
<macro dprint 253d>
void    _dprint(char *, ...);
void    *eamalloc(ulong, ulong);
void    *emalloc(ulong);
void    *earealloc(void *, ulong, ulong);
void    *erealloc(void *, ulong);
char    *estrdup(char *);
int    slurpdir(char *, Dir **);
errorneg1 hparse(Hash *, char *);
char    *strip(char *);
int    showprogress(int, int);
u64int  murmurhash2(void*, usize);
Qid    parseqid(char*);
int    charval(int);

/* packing */
void    dtinit(Dtab *, Object*);
void    dtclear(Dtab*);
Delta*  deltify(Object*, Dtab*, int*);

/* proto handling */
int    readpkt(Conn*, char*, int);
int    writepkt(Conn*, char*, int);
int    fmtpkt(Conn*, char*, ...);
int    flushpkt(Conn*);
void    initconn(Conn*, int, int);
int    gitconnect(Conn *, char *, char *);
int    readphase(Conn *);
int    writephase(Conn *);
void    closeconn(Conn *);
void    parsecaps(char *, Conn *);
int    okref(char*);

/* queues */
void    qinit(Objq*);
void    qclear(Objq*);
void    qput(Objq*, Object*, int);
int    qpop(Objq*, Qelt*);

```

Uses Buf 270, Cinfo 270, Conn 270, Dblock 270, Delta 270, Dirent 270, Dtab 270, Hash 270, KiB 270, Object 270, Objlist 270, Objq 270, Objset 270, Qelt 270, and Tinfo 270.

D.1.23 git9/log.c

```

<git9/log.c 272>≡
#include <u.h>
#include <libc.h>
#include "git.h"

```

```

typedef struct Pfilt Pfilt;
<struct Pfilt 143a>

<global out(log.c) 140c>
<global queryexpr(log.c) 146a>
<global commitid(log.c) 140b>
<global shortlog(log.c) 145b>
<global msgcount(log.c) 141b>

<global done(log.c) 141e>
<global objq(log.c) 141d>

<global pathfilt(log.c) 143b>

<function filteradd 143e>

<function lookup (git9/log.c) 145a>

<function matchesfilter1 144b>
<function matchesfilter 144a>

<function nextline 259a>

<function show 142b>
<function showquery 146d>
<function showcommits 141f>

<function usage (git9/log.c) 140a>
<function main (git9/log.c) 140d>

```

Uses Pfilt, out 143a, and readobject() 157b.

D.1.24 git9/objset.c

```

<git9/objset.c 273a>≡
#include <u.h>
#include <libc.h>

#include "git.h"

<function osinit 37c>
<function osclear 37d>

<function osadd 37e>
<function osfind 38c>
<function oshas 38d>

```

D.1.25 git9/ols.c

```

<function crackidx 273b>≡ (276a)
static int
crackidx(char *path, int *np)
{
    int fd;
    char buf[4];

    if((fd = open(path, OREAD)) == -1)
        return -1;

```

```

if(seek(fd, 8 + 255*4, 0) == -1)
    return -1;
if(readn(fd, buf, sizeof(buf)) != sizeof(buf))
    return -1;
*np = GETBE32(buf);
return fd;
}

```

Uses GETBE32.

```

⟨function isloosedir 274a⟩≡ (276a)
int
isloosedir(char *s)
{
    return strlen(s) == 2 && isxdigit(s[0]) && isxdigit(s[1]);
}

```

```

⟨function olsreadpacked 274b⟩≡ (276a)
int
olsreadpacked(Objlist *ols, Hash *h)
{
    char *p;
    int i, j;

    i = ols->packidx;
    j = ols->entidx;

    if(ols->state == Siter)
        goto step;
    for(i = 0; i < ols->npack; i++){
        if(!endswith(ols->pack[i].name, ".idx"))
            continue;
        if((p = smprint(".git/objects/pack/%s", ols->pack[i].name)) == nil)
            sysfatal("smprint: %r");
        ols->fd = crackidx(p, &ols->nent);
        free(p);
        if(ols->fd == -1)
            continue;
        j = 0;
        while(j < ols->nent){
            if(readn(ols->fd, h->h, sizeof(h->h)) != sizeof(h->h))
                continue;
            ols->state = Siter;
            ols->packidx = i;
            ols->entidx = j;
            return 0;
        }
    }
step:
    j++;
}
close(ols->fd);
}
ols->state = Sinit;
return -1;
}

```

Uses Sinit-28 276a, Siter-29 276a, crackidx() 273b, and endswith() 259c.

```

⟨function olsreadloose 274c⟩≡ (276a)
int
olsreadloose(Objlist *ols, Hash *h)
{
    char buf[64], *p;
}

```

```

int i, j, n;

i = ols->topidx;
j = ols->looseidx;
if(ols->state == Siter)
    goto step;
for(i = 0; i < ols->ntop; i++){
    if(!isloosedir(ols->top[i].name))
        continue;
    if((p = smprint(".git/objects/%s", ols->top[i].name)) == nil)
        sysfatal("smprint: %r");
    ols->fd = open(p, OREAD);
    free(p);
    if(ols->fd == -1)
        continue;
    while((ols->nloose = dirread(ols->fd, &ols->loose)) > 0){
        j = 0;
        while(j < ols->nloose){
            n = snprintf(buf, sizeof(buf), "%s%s", ols->top[i].name, ols->loose[j].name);
            if(n >= sizeof(buf))
                goto step;
            if(hparse(h, buf) == ERROR_NEG1)
                goto step;
            ols->state = Siter;
            ols->topidx = i;
            ols->looseidx = j;
            return 0;
        }
    }
step:
    j++;
}
free(ols->loose);
ols->loose = nil;
}
close(ols->fd);
ols->fd = -1;
}
ols->state = Sinit;
return -1;
}

```

Uses Sinit-28 276a, Siter-29 276a, hparse() 31d, and isloosedir() 274a.

<function olsnext 275>≡ (276a)

```

int
olsnext(Objlist *ols, Hash *h)
{
    if(ols->stage == 0){
        if(olsreadloose(ols, h) != -1){
            ols->idx++;
            return 0;
        }
        ols->stage++;
    }
    if(ols->stage == 1){
        if(olsreadpacked(ols, h) != -1){
            ols->idx++;
            return 0;
        }
        ols->stage++;
    }
    return -1;
}

```

```
}
```

Uses `olsreadloose()` 274c and `olsreadpacked()` 274b.

```
<git9/ols.c 276a>≡  
#include <u.h>  
#include <libc.h>  
#include <ctype.h>  
#include "git.h"  
  
enum {  
    Sinit,  
    Siter,  
};  
  
<function crackidx 273b>  
  
<function isloosedir 274a>  
  
<function endswith 259c>  
  
<function olsreadpacked 274b>  
  
<function olsreadloose 274c>  
  
<function mkols 77a>  
  
<function olsfree 68b>  
  
<function olsnext 275>
```

D.1.26 git9/pack.c

```
<function crc32 276b>≡ (277)  
static u32int  
crc32(u32int crc, char *b, int nb)  
{  
    static u32int crctab[256] = {  
        0x00000000, 0x77073096, 0xee0e612c, 0x990951ba, 0x076dc419, 0x706af48f, 0xe963a535,  
        0x9e6495a3, 0x0edb8832, 0x79dcb8a4, 0xe0d5e91e, 0x97d2d988, 0x09b64c2b, 0x7eb17cbd,  
        0xe7b82d07, 0x90bf1d91, 0x1db71064, 0x6ab020f2, 0xf3b97148, 0x84be41de, 0x1adad47d,  
        0x6ddde4eb, 0xf4d4b551, 0x83d385c7, 0x136c9856, 0x646ba8c0, 0xfd62f97a, 0x8a65c9ec,  
        0x14015c4f, 0x63066cd9, 0xfa0f3d63, 0x8d080df5, 0x3b6e20c8, 0x4c69105e, 0xd56041e4,  
        0xa2677172, 0x3c03e4d1, 0x4b04d447, 0xd20d85fd, 0xa50ab56b, 0x35b5a8fa, 0x42b2986c,  
        0xdbbbc9d6, 0xacbcf940, 0x32d86ce3, 0x45df5c75, 0xdcd60dcf, 0xabd13d59, 0x26d930ac,  
        0x51de003a, 0xc8d75180, 0xbfd06116, 0x21b4f4b5, 0x56b3c423, 0xcfba9599, 0xb8bda50f,  
        0x2802b89e, 0x5f058808, 0xc60cd9b2, 0xb10be924, 0x2f6f7c87, 0x58684c11, 0xc1611dab,  
        0xb6662d3d, 0x76dc4190, 0x01db7106, 0x98d220bc, 0xefd5102a, 0x71b18589, 0x06b6b51f,  
        0x9fbfe4a5, 0xe8b8d433, 0x7807c9a2, 0x0f00f934, 0x9609a88e, 0xe10e9818, 0x7f6a0dbb,  
        0x086d3d2d, 0x91646c97, 0xe6635c01, 0x6b6b51f4, 0x1c6c6162, 0x856530d8, 0xf262004e,  
        0x6c0695ed, 0x1b01a57b, 0x8208f4c1, 0xf50fc457, 0x65b0d9c6, 0x12b7e950, 0x8bbeb8ea,  
        0xfcb9887c, 0x62dd1ddf, 0x15da2d49, 0x8cd37cf3, 0xfbd44c65, 0x4db26158, 0x3ab551ce,  
        0xa3bc0074, 0xd4bb30e2, 0x4adfa541, 0x3dd895d7, 0xa4d1c46d, 0xd3d6f4fb, 0x4369e96a,  
        0x346ed9fc, 0xad678846, 0xda60b8d0, 0x44042d73, 0x33031de5, 0xaa0a4c5f, 0xdd0d7cc9,  
        0x5005713c, 0x270241aa, 0xbe0b1010, 0xc90c2086, 0x5768b525, 0x206f85b3, 0xb966d409,  
        0xce61e49f, 0x5edef90e, 0x29d9c998, 0xb0d09822, 0xc7d7a8b4, 0x59b33d17, 0x2eb40d81,  
        0xb7bd5c3b, 0xc0ba6cad, 0xedb88320, 0x9abfb3b6, 0x03b6e20c, 0x74b1d29a, 0xeada54739,  
        0x9dd277af, 0x04db2615, 0x73dc1683, 0xe3630b12, 0x94643b84, 0x0d6d6a3e, 0x7a6a5aa8,  
        0xe40ecf0b, 0x9309ff9d, 0x0a00ae27, 0x7d079eb1, 0xf00f9344, 0x8708a3d2, 0x1e01f268,  
        0x6906c2fe, 0xf762575d, 0x806567cb, 0x196c3671, 0x6e6b06e7, 0xfed41b76, 0x89d32be0,
```

```

0x10da7a5a, 0x67dd4acc, 0xf9b9df6f, 0x8ebeeef9, 0x17b7be43, 0x60b08ed5, 0xd6d6a3e8,
0xa1d1937e, 0x38d8c2c4, 0x4fdff252, 0xd1bb67f1, 0xa6bc5767, 0x3fb506dd, 0x48b2364b,
0xd80d2bda, 0xaf0a1b4c, 0x36034af6, 0x41047a60, 0xdf60efc3, 0xa867df55, 0x316e8eef,
0x4669be79, 0xcb61b38c, 0xbc66831a, 0x256fd2a0, 0x5268e236, 0xcc0c7795, 0xbb0b4703,
0x220216b9, 0x5505262f, 0xc5ba3bbe, 0xb2bd0b28, 0x2bb45a92, 0x5cb36a04, 0xc2d7ffa7,
0xb5d0cf31, 0x2cd99e8b, 0x5bdeae1d, 0x9b64c2b0, 0xec63f226, 0x756aa39c, 0x026d930a,
0x9c0906a9, 0xeb0e363f, 0x72076785, 0x05005713, 0x95bf4a82, 0xe2b87a14, 0x7bb12bae,
0x0cb61b38, 0x92d28e9b, 0xe5d5be0d, 0x7cdcefb7, 0x0bdbdf21, 0x86d3d2d4, 0xfd4e242,
0x68ddb3f8, 0x1fda836e, 0x81be16cd, 0xf6b9265b, 0x6fb077e1, 0x18b74777, 0x88085ae6,
0xff0f6a70, 0x66063bca, 0x11010b5c, 0x8f659eff, 0xf862ae69, 0x616bffd3, 0x166ccf45,
0xa00ae278, 0xd70dd2ee, 0x4e048354, 0x3903b3c2, 0xa7672661, 0xd06016f7, 0x4969474d,
0x3e6e77db, 0xaed16a4a, 0xd9d65adc, 0x40df0b66, 0x37d83bf0, 0xa9bcae53, 0xdebb9ec5,
0x47b2cf7f, 0x30b5ffe9, 0xbdbdf21c, 0xcabac28a, 0x53b39330, 0x24b4a3a6, 0xbad03605,
0xcdd70693, 0x54de5729, 0x23d967bf, 0xb3667a2e, 0xc4614ab8, 0x5d681b02, 0x2a6f2b94,
0xb40bbe37, 0xc30c8ea1, 0x5a05df1b, 0x2d02ef8d

```

```

};
int i;

crc ^= 0xFFFFFFFF;
for(i = 0; i < nb; i++)
    crc = (crc >> 8) ^ crctab[(crc ^ b[i]) & 0xFF];
return crc ^ 0xFFFFFFFF;
}

```

Uses `bappend()` 160e, `bdecompress()` 276b, and `breadc()` 276b.

```

<git9/pack.c 277>≡
#include <u.h>
#include <libc.h>
#include <ctype.h>

#include "git.h"

typedef struct Buf Buf;
typedef struct Metavec Metavec;
typedef struct Meta Meta;
typedef struct Compout Compout;
typedef struct Packf Packf;

<macro max 256a>

<struct Metavec 153c>

<struct Meta 152d>

<struct Compout 171b>

<struct Buf 46c>

<struct Packf 152a>

static int readpacked(Biobuf *, Object *, int);
static Object *readidxobject(Biobuf *, Hash, int);

<global objcache 248a>
<globals lruxxx 249b>

vlong cachemax = 128*MiB;

<globals packf 152b>
<global openpacks 161b>

```

<function clear 34a>
<function unref 33f>
<function ref 33e>
<function cache 249d>
<function loadpack 159c>
<function refreshpacks 158d>
<function openpack 160d>
<function closepack 160e>
<function crc32 276b>
<function bappend 47a>
<function breadc 46e>
<function bdecompress 46d>
<function decompress 46b>
<function readvint 179a>
<function applydelta 178>
<function readrdelta 180e>
<function readodelta 177d>
<function readpacked 161e>
<function readloose 44>
<function hashcmp 30d>
<function searchindex 155c>
<function scanword 259b>
<function nextline (git9/pack.c) 258d>
<function parseauthor 49b>
<function parsecommit 48c>
<function parsetree 47c>
<function parsetag 238f>
<function parseobject 45d>
<function readidxobject 43b>
<function expandprefix 52a>
<function readobject 43a>
<function clearedobject 250>
<function objcmp 32c>

<function hwrite 176b>
 <function objectcrc 176d>
 <function indexpack 173b>
 <function deltaordercmp 153b>
 <function writeordercmp 152e>
 <function addmeta 165a>
 <function freemeta 153a>
 <function loadtree 165b>
 <function loadcommit 164d>
 <function readmeta 163c>
 <function deltasz 167a>
 <function pickdeltas 166>
 <function compread 172a>
 <function compwrite 172b>
 <function hcompress 171c>
 <function append 173a>
 <function encodedelta 172c>
 <function packhdr 170b>
 <function packoff 171a>
 <function genpack 169b>
 <function writepack 163b>

Uses Ccache 33c, Cexist, Cidx 33c, Compout, GCommit 33a, Meta 153c, Metavec, Packf 171b, compread() 163c, compwrite() 163c, parseobject() 48c, readpacked() 180e, and ref() 36c.

D.1.27 git9/proto.c

```

<git9/proto.c 279>≡
#include <u.h>
#include <libc.h>
#include <ctype.h>

#include "git.h"

<constant Useragent 244a>
<constant Contenthdr 245a>
<constant Accepthdr 245b>

enum {
    Nproto = 16,
  
```

```
Nport    = 16,  
Nhost    = 256,  
Npath    = 128,  
Nbranch  = 32,  
};
```

<function matchcap 210c>

<function parsecaps 210b>

<function tracepkt 254f>

<function readpkt 209c>

<function writepkt 208e>

<function fmtpkt 209b>

<function flushpkt 209a>

<function grab 207a>

<function parseuri 206c>

<function webclone 243b>

<function webopen 243a>

<function issmarthttp 244b>

<function dialhttp 242g>

<function dialssh 246b>

<function githandshake 208d>

<function dialhgit 247b>

<function initconn 212b>

<function dialgit 207b>

<function servelocal 186a>

<function localrepo 185d>

<function gitconnect 185b>

<function writephase 245c>

<function readphase 245d>

<function closeconn 185c>

Uses dialgit() 212b.

D.1.28 git9/query.c

<git9/query.c 280>≡

```
#include <u.h>
```

```
#include <libc.h>
```

```
#include "git.h"
```

```
#pragma varargck    type    "P" void
```

```

// flags
⟨global fullpath(query.c) 101a⟩
⟨global changes(query.c) 98b⟩
⟨global reverse(query.c) 101f⟩

⟨global path(query.c) 100c⟩
⟨global npath(query.c) 100d⟩
⟨function Pfmt 100b⟩

⟨function showdir 100f⟩
⟨function show (git9/query.c) 100e⟩

⟨function difftrees 99⟩
⟨function diffcommits 98e⟩

⟨function usage (git9/query.c) 87a⟩
⟨function main (git9/query.c) 87b⟩

```

D.1.29 git9/ref.c

```

⟨git9/ref.c 281⟩≡
#include <u.h>
#include <libc.h>
#include <ctype.h>

#include "git.h"

typedef struct Eval Eval;

⟨enum PaintQcolor 95e⟩
⟨enum PaintMode 95d⟩

⟨struct Eval 88d⟩

⟨constant colors 98a⟩

⟨global zcommit(ref.c) 37a⟩

⟨function eatspace 91a⟩

⟨function push 92c⟩
⟨function pop 92e⟩

⟨function isword 92b⟩
⟨function word 92a⟩

⟨function take 90a⟩

⟨function paint 96⟩

⟨function findtwixt 164a⟩

⟨function ancestor 94a⟩

⟨function lca 93d⟩

⟨function parent 93a⟩

```

<function range 94c>
<function matchpfx 51f>
<function readref (git9/ref.c) 50c>
<function evalpostfix 90b>
<function evalexpr 89c>
<function resolverefs 89a>
<function resolveref 89b>
<function readrefdir 150e>
<function listrefs 150d>

Uses Eval 88d.

D.1.30 git9/repack.c

```
<git9/repack.c 282a>≡  
#include <u.h>  
#include <libc.h>  
  
#include "git.h"  
  
<macro TMPPATH 150a>  
  
<function cleanup 151>  
  
<function usage (git9/repack.c) 149a>  
<function main (git9/repack.c) 149b>
```

D.1.31 git9/save.c

```
<git9/save.c 282b>≡  
#include <u.h>  
#include <libc.h>  
#include "git.h"  
  
typedef struct Objbuf Objbuf;  
  
<struct Objbuf 55a>  
  
enum {  
    <const Maxparents 58c>  
};  
  
<globals authorxxx(save.c) 115b>  
<globals comitterxxx(save.c) 115c>  
<global commitmsg(save.c) 115d>  
<globals parents(save.c) 58b>  
  
<globals idx(save.c) 41d>  
  
<function gitmode 58a>  
  
<function namecmp 117b>  
  
<function idxcmp 41b>
```

<function bwrite 56b>
<function objbytes 56c>

<function writeobj 55b>

<function writetree 57c>

<function blobify 56d>

<function tracked 120a>

<function dirent 119>

<function treeify 118>

<function mkcommit 58d>

<function findroot 117f>

<function usage (git9/save.c) 115a>
<function main (git9/save.c) 115e>

Uses GTree 33a, Objbuf 282b, bwrite(), and objbytes().

D.1.32 git9/send.c

```
<git9/send.c 283a>≡  
#include <u.h>  
#include <libc.h>  
  
#include "git.h"  
  
typedef struct Map Map;  
  
<struct Map 198c>  
  
<global sendall(send.c) 202e>  
<global force(send.c) 202a>  
<globals branch(send.c) 203b>  
<globals removed(send.c) 202c>  
  
<function findref 201a>  
<function findkey 198d>  
  
<function readours 201b>  
  
<function sendpack 199>  
  
<function usage (git9/send.c) 198a>  
<function main (git9/send.c) 198b>
```

Uses Map and Pktmax.

D.1.33 git9/serve.c

```
<git9/serve.c 283b>≡  
#include <u.h>
```

```

#include <libc.h>
#include <ctype.h>
#include <auth.h>

#include "git.h"

<global pathpfx(serve.c) 222a>
<global allowwrite(serve.c) 211b>

<function fail (git9/serve.c) 212d>

<function gethead 215a>

<function showrefs 214>

<function servnegotiate 213b>

<function servpack 213a>

<function validref 217a>

<function recvnegotiate 216>

<function rename (git9/serve.c) 218b>
<function checkhash (git9/serve.c) 219a>

<function mkdir 218a>
<function mkpath 221b>

<function updatepack 217b>

<function lockrepo 221a>

<function updaterefs 219b>

<function recvpack 215b>

<function parsecmd 212c>

<function usage (git9/serve.c) 211a>
<function main (git9/serve.c) 211c>

```

Uses Pktmax.

D.1.34 git9/util.c

```

<global interactive 284a>≡ (284b)
    bool interactive = true;

<git9/util.c 284b>≡
#include <u.h>
#include <libc.h>
#include <ctype.h>

#include "git.h"

Reprog *authorpat;
<global Zhash 30b>
<global chattygit 253c>
<global interactive 284a>

```

```

<global gitdirmode 61g>

enum {
    Seed          = 2928213749ULL
};

<function emptydir 35c>
<function entcmp 35b>

<function hasheq 30c>

<function charval 31e>

<function emalloc 256b>
<function eamalloc 256c>
<function erealloc 256d>
<function earealloc 257a>
<function estrdup 257b>

<function Hfmt 31c>
<function Tfmt 252c>
<function Ofmt 252d>
<function Qfmt 253a>

<function findrepo 62a>
<function gitinit 61e>

<function hparse 31d>

<function slurpdir 260>

<function strip 258c>

<function _dprint 253e>

<function showprogress 176c>

<function qinit 39b>
<function qclear 39c>
<function qput 39d>
<function qpop 40a>

<function murmurhash2 224>

<function parseqid 53c>

<function okref 191>

```

D.1.35 git9/walk.c

```

<git9/walk.c 285>≡
#include <u.h>
#include <libc.h>
#include "git.h"

typedef struct Idxed    Idxed;
typedef struct Idxent   Idxent;

<enum WalkFlags 40b>

```

<global repopath(walk.c) 102b>
<global wdirpath(walk.c) 102c>

<global relapath(walk.c) 113c>
<global nslash(walk.c) 113d>

<globals xxxstr 108b>

<constant bdir 109e>

<global nrel(walk.c) 102f>
<global quiet(walk.c) 103b>

<global dirty(walk.c) 112c>

<global isindexed(walk.c) 102d>
<global intree(walk.c) 104b>
<global printflg(walk.c) 40c>

<globals idx(walk.c) 42b>

<global cleanidx 109d>
<global staleidx(walk.c) 102e>

<globals wdir(walk.c) 104a>

`void loadwdir(char*);`

<function checkedin 109f>

<function pathcmp 106c>

<function indexed 106b>

<function idxcmp (git9/walk.c) 41c>

<function samedata 110>

<function loadent 105a>
<function loadwdir 104d>

<function pfxmatch 108a>

<function reporel 106a>

<function show (git9/walk.c) 109c>

<function findslashes 113e>

<function usage (git9/walk.c) 102a>
<function main (git9/walk.c) 102g>

Uses `Idxed` 102c, `Idxent` 270, `cleanidx` 285, `isindexed` 285, and `ustr` 285.

D.2 diff/

D.2.1 diff/diff.c

```
<diff/diff.c 287a>≡  
#include <u.h>  
#include <libc.h>  
#include <bio.h>  
  
#include "diff.h"  
  
<function done(diff) 227a>  
  
<function usage(diff) 226a>  
  
<function main(diff) 226b>
```

D.2.2 diff/diff.h

```
<constant MAXPATHLEN(diff) 287b>≡ (287f)  
#define MAXPATHLEN 1024  
  
<constant MAXLINELEN(diff) 287c>≡ (287f)  
#define MAXLINELEN 4096  
  
<macro DIRECTORY(diff) 287d>≡ (287f)  
#define DIRECTORY(s) ((s)->qid.type&QTDIR)  
  
<macro REGULAR_FILE(diff) 287e>≡ (287f)  
#define REGULAR_FILE(s) ((s)->type == 'M' && !DIRECTORY(s))  
  
<diff/diff.h 287f>≡  
typedef struct Line Line;  
typedef struct Cand Cand;  
typedef struct Diff Diff;  
typedef struct Change Change;  
  
struct Line {  
    int serial;  
    int value;  
};  
  
struct Cand {  
    int x;  
    int y;  
    int pred;  
};  
  
struct Change  
{  
    int oldx;  
    int oldy;  
    int newx;  
    int newy;  
};  
  
struct Diff {  
    Cand cand;  
    Line *file[2], line;
```

```

int len[2];
int binary;
int bindiff;
Line *sfile[2]; /*shortened by pruning common prefix and suffix*/
int slen[2];
int pref, suff; /*length of prefix and suffix*/
int *class; /*will be overlaid on file[0]*/
int *member; /*will be overlaid on file[1]*/
int *klist; /*will be overlaid on file[0] after class*/
Cand *clist; /* merely a free storage pot for candidates */
int clen;
int *J; /*will be overlaid on class*/
long *ixold; /*will be overlaid on klist*/
long *ixnew; /*will be overlaid on file[1]*/
char *file1;
char *file2;
Biobuf *input[2];
Biobuf *b0;
Biobuf *b1;
int firstchange;
Change *changes;
int nchanges;
};

```

```

extern char mode;
extern char bflag;
extern char rflag;
extern char mflag;
extern int anychange;
extern Biobuf stdout;

```

<constant MAXPATHLEN(diff) 287b>
<constant MAXLINELEN(diff) 287c>

<macro DIRECTORY(diff) 287d>
<macro REGULAR_FILE(diff) 287e>

```

int mkpathname(char *, char *, char *);
char *mktmpfile(int, Dir **);
char *statfile(char *, Dir **);
void *emalloc(unsigned);
void *erealloc(void *, unsigned);
void diff(char *, char *, int);
void diffreg(char*, char*, char*, char*);
void diffdir(char *, char *, int);
void calcdiff(Diff *, char *, char *, char *, char *);
Biobuf *prepare(Diff*, int, char *, char *);
void check(Diff *, Biobuf *, Biobuf *);
void change(Diff *, int, int, int, int);
void freediff(Diff *);
void flushchanges(Diff *);
void fetch(Diff *d, long *f, int a, int b, Biobuf *bp, char *s);
int readline(Biobuf*, char*, int);

```

Uses Cand 287f, Change 287f, Diff 287f, and Line 287f.

D.2.3 diff/diffdir.c

<function itemcmp(diff) 288>≡
static int

(291a)

```

itemcmp(void *v1, void *v2)
{
    char **d1 = v1, **d2 = v2;

    return strcmp(*d1, *d2);
}

```

<function scandir(diff) 289a> ≡ (291a)

```

static char **
scandir(char *name)
{
    char **cp;
    Dir *db;
    int nitems;
    int fd, n;

    if ((fd = open(name, OREAD)) < 0) {
        fprintf(2, "%s: can't open %s: %r\n", argv0, name);
        /* fake an empty directory */
        cp = emalloc(sizeof(char*));
        cp[0] = 0;
        return cp;
    }
    cp = 0;
    nitems = 0;
    if((n = dirreadall(fd, &db)) > 0){
        while (n--) {
            cp = erealloc(cp, (nitems+1)*sizeof(char*));
            cp[nitems] = emalloc(strlen((db+n)->name)+1);
            strcpy(cp[nitems], (db+n)->name);
            nitems++;
        }
        free(db);
    }
    cp = erealloc(cp, (nitems+1)*sizeof(char*));
    cp[nitems] = 0;
    close(fd);
    qsort((char *)cp, nitems, sizeof(char*), itemcmp);
    return cp;
}

```

Uses `itemcmp()` 288.

<function isdotordotdot(diff) 289b> ≡ (291a)

```

static int
isdotordotdot(char *p)
{
    if (*p == '.') {
        if (!p[1])
            return 1;
        if (p[1] == '.' && !p[2])
            return 1;
    }
    return 0;
}

```

<function diffdir(diff) 289c> ≡ (291a)

```

void
diffdir(char *f, char *t, int level)
{
    char **df, **dt, **dirf, **dirt;
}

```

```

char *from, *to;
int res;
char fb[MAXPATHLEN+1], tb[MAXPATHLEN+1];

df = scandir(f);
dt = scandir(t);
dirf = df;
dirt = dt;
while (*df || *dt) {
    from = *df;
    to = *dt;
    if (from && isdotordotdot(from)) {
        df++;
        continue;
    }
    if (to && isdotordotdot(to)) {
        dt++;
        continue;
    }
    if (!from)
        res = 1;
    else if (!to)
        res = -1;
    else
        res = strcmp(from, to);
    if (res < 0) {
        if (mode == 0 || mode == 'n')
            Bprintf(&stdout, "Only in %s: %s\n", f, from);
        df++;
        continue;
    }
    if (res > 0) {
        if (mode == 0 || mode == 'n')
            Bprintf(&stdout, "Only in %s: %s\n", t, to);
        dt++;
        continue;
    }
    if (mkpathname(fb, f, from))
        continue;
    if (mkpathname(tb, t, to))
        continue;
    diff(fb, tb, level+1);
    df++; dt++;
}
for (df = dirf; *df; df++)
    free(*df);
for (dt = dirt; *dt; dt++)
    free(*dt);
free(dirf);
free(dirt);
}

```

Uses MAXPATHLEN 287b, diff() 290, isdotordotdot() 289b, mkpathname() 310b, mode 311b, scandir() 289a, and stdout 311b.

<function diff(diff) 290> ≡ (291a)

```

void
diff(char *f, char *t, int level)
{
    char *fp, *tp, *p, fb[MAXPATHLEN+1], tb[MAXPATHLEN+1];
    Dir *fsb, *tsb;

```

```

fsb = nil;
tsb = nil;
if ((fp = statfile(f, &fsb)) == 0)
    goto Return;
if ((tp = statfile(t, &tsb)) == 0)
    goto Return;
if (DIRECTORY(fsb) && DIRECTORY(tsb)) {
    if (rflag || level == 0)
        diffdir(fp, tp, level);
    else
        Bprint(&stdout, "Common subdirectories: %s and %s\n", fp, tp);
}
else if (REGULAR_FILE(fsb) && REGULAR_FILE(tsb))
    diffreg(fp, f, tp, t);
else {
    if (REGULAR_FILE(fsb)) {
        if ((p = utfrrune(f, '/')) == 0)
            p = f;
        else
            p++;
        if (mkpathname(tb, tp, p) == 0)
            diffreg(fp, f, tb, t);
    } else {
        if ((p = utfrrune(t, '/')) == 0)
            p = t;
        else
            p++;
        if (mkpathname(fb, fp, p) == 0)
            diffreg(fb, f, tp, t);
    }
}
}
Return:
    free(fsb);
    free(tsb);
}

```

Uses `DIRECTORY` 287d, `MAXPATHLEN` 287b, `REGULAR_FILE` 287e, `diffdir()` 289c, `diffreg()` 304b, `mkpathname()` 310b, `rflag` 311b, `statfile()` 311a, and `stdout` 311b.

`<diff/diffdir.c 291a>`≡

```

#include <u.h>
#include <libc.h>
#include <bio.h>
#include "diff.h"

<function itemcmp(diff) 288>

<function scandir(diff) 289a>

<function isdotordotdot(diff) 289b>

<function diffdir(diff) 289c>

<function diff(diff) 290>

```

D.2.4 diff/diffio.c

`<macro MIN(diff) 291b>`≡ (298)
`#define MIN(x, y) ((x) < (y) ? (x) : (y))`

<function readline(diff) 292a>≡ (298)

```
int
readline(Biobuf *bp, char *buf, int nbuf)
{
    int c;
    char *p, *e;

    p = buf;
    e = p + nbuf-1;
    do {
        c = Bgetc(bp);
        if (c < 0) {
            if (p == buf)
                return -1;
            break;
        }
        *p++ = c;
        if (c == '\n')
            break;
    } while (p < e);
    *p = 0;
    if (c != '\n' && c >= 0) {
        do c = Bgetc(bp);
        while (c >= 0 && c != '\n');
    }
    return p - buf;
}
```

<constant HALFLONG(diff) 292b>≡ (298)

```
#define HALFLONG 16
```

<macro low(diff) 292c>≡ (298)

```
#define low(x) (x&((1L<<HALFLONG)-1))
```

<macro high(diff) 292d>≡ (298)

```
#define high(x) (x>>HALFLONG)
```

<function readhash(diff) 292e>≡ (298)

```
/*
 * hashing has the effect of
 * arranging line in 7-bit bytes and then
 * summing 1-s complement in 16-bit hunks
 */
static int
readhash(Biobuf *bp, char *buf, int nbuf)
{
    long sum;
    unsigned shift;
    char *p;
    int len, space;

    sum = 1;
    shift = 0;
    if ((len = readline(bp, buf, nbuf)) == -1)
        return 0;
    p = buf;
    switch(bflag) /* various types of white space handling */
    {
    case 0:
        while (len--) {
```

```

        sum += (long)*p++ << (shift &= (HALFLONG-1));
        shift += 7;
    }
    break;
case 1:
    /*
     * coalesce multiple white-space
     */
    for (space = 0; len--; p++) {
        if (isspace(*p)) {
            space++;
            continue;
        }
        if (space) {
            shift += 7;
            space = 0;
        }
        sum += (long)*p << (shift &= (HALFLONG-1));
        shift += 7;
    }
    break;
default:
    /*
     * strip all white-space
     */
    while (len--) {
        if (isspace(*p)) {
            p++;
            continue;
        }
        sum += (long)*p++ << (shift &= (HALFLONG-1));
        shift += 7;
    }
    break;
}
sum = low(sum) + high(sum);
return ((short)low(sum) + (short)high(sum));
}

```

Uses HALFLONG-4 292b, bflag 311b, high-6 292d, and low-5 292c.

<function prepare(diff) 293 ≡ (298)

```

Biobuf *
prepare(Diff *d, int i, char *arg, char *orig)
{
    Line *p;
    int j, h;
    Biobuf *bp;
    char *cp, buf[MAXLINELEN];
    int nbytes;
    Rune r;

    if (i == 0) {
        d->file1 = orig;
        d->firstchange = 0;
    } else
        d->file2 = orig;
    bp = Bopen(arg, OREAD);
    if (!bp)
        sysfatal("cannot open %s: %r", arg);
    if (d->binary)

```

```

    return bp;
nbytes = Bread(bp, buf, MIN(1024, MAXLINELEN));
if (nbytes > 0) {
    cp = buf;
    while (cp < buf+nbytes-UTFmax) {
        /*
         * heuristic for a binary file in the
         * brave new UNICODE world
         */
        cp += chartorune(&r, cp);
        if (r == 0 || (r > 0x7f && r <= 0xa0)) {
            d->binary++;
            return bp;
        }
    }
    Bseek(bp, 0, 0);
}
p = emalloc(3*sizeof(Line));
for (j = 0; h = readhash(bp, buf, sizeof(buf)); p[j].value = h)
    p = erealloc(p, (++j+3)*sizeof(Line));
d->len[i] = j;
d->file[i] = p;
d->input[i] = bp;
return bp;
}

```

Uses MAXLINELEN 287c, MIN-3 291b, and readhash() 292e.

<function squishspace(diff) 294a> ≡ (298)

```

static int
squishspace(char *buf)
{
    char *p, *q;
    int space;

    for (space = 0, q = p = buf; *q; q++) {
        if (isspace(*q)) {
            space++;
            continue;
        }
        if (space && bflag == 1) {
            *p++ = ' ';
            space = 0;
        }
        *p++ = *q;
    }
    *p = 0;
    return p - buf;
}

```

Uses bflag 311b.

<function check(diff) 294b> ≡ (298)

```

/*
 * need to fix up for unexpected EOF's
 */
void
check(Diff *d, Biobuf *bf, Biobuf *bt)
{
    int f, t, flen, tlen;
    char fbuf[MAXLINELEN], tbuf[MAXLINELEN];
}

```

```

d->ixold[0] = 0;
d->ixnew[0] = 0;
for (f = t = 1; f < d->len[0]; f++) {
    flen = readline(bf, fbuf, sizeof(fbuf));
    d->ixold[f] = d->ixold[f-1] + flen; /* ftell(bf) */
    if (d->J[f] == 0)
        continue;
    do {
        tlen = readline(bt, tbuf, sizeof(tbuf));
        d->ixnew[t] = d->ixnew[t-1] + tlen; /* ftell(bt) */
    } while (t++ < d->J[f]);
    if (bflag) {
        flen = squishspace(fbuf);
        tlen = squishspace(tbuf);
    }
    if (flen != tlen || strcmp(fbuf, tbuf))
        d->J[f] = 0;
}
while (t < d->len[1]) {
    tlen = readline(bt, tbuf, sizeof(tbuf));
    d->ixnew[t] = d->ixnew[t-1] + tlen; /* fseek(bt) */
    t++;
}
}

```

Uses MAXLINELEN 287c, bflag 311b, and squishspace() 294a.

```

⟨function range(diff) 295a⟩≡ (298)
static void
range(int a, int b, char *separator)
{
    Bprint(&stdout, "%d", a > b ? b : a);
    if (a < b)
        Bprint(&stdout, "%s%d", separator, b);
}

```

Uses stdout 311b.

```

⟨function fetch(diff) 295b⟩≡ (298)
void
fetch(Diff *d, long *f, int a, int b, Biobuf *bp, char *s)
{
    char buf[MAXLINELEN];
    int maxb, len;

    if(a <= 1)
        a = 1;
    if(bp == d->input[0])
        maxb = d->len[0];
    else
        maxb = d->len[1];
    if(b > maxb)
        b = maxb;
    if(a > maxb)
        return;
    Bseek(bp, f[a-1], 0);
    while (a++ <= b) {
        len = readline(bp, buf, sizeof(buf));
        if(len == 0 || buf[len-1] != '\n'){
            Bprint(&stdout, "%s%s\n", s, buf);
            Bprint(&stdout, "\\ No newline at end of file\n");
        }else
    }
}

```

```

        Bprint(&stdout, "%s%s", s, buf);
    }
}

```

Uses MAXLINELEN 287c and stdout 311b.

```

⟨function change(diff) 296⟩≡ (298)
void
change(Diff *df, int a, int b, int c, int d)
{
    char verb;
    char buf[4];
    Change *ch;

    if (a > b && c > d)
        return;
    anychange = 1;
    if (mflag && df->firstchange == 0) {
        if(mode) {
            buf[0] = '-';
            buf[1] = mode;
            buf[2] = ' ';
            buf[3] = '\0';
        } else {
            buf[0] = '\0';
        }
        Bprint(&stdout, "diff %s%s %s\n", buf, df->file1, df->file2);
        df->firstchange = 1;
    }
    verb = a > b ? 'a': c > d ? 'd': 'c';
    switch(mode) {
    case 'e':
        range(a, b, ",");
        Bputc(&stdout, verb);
        break;
    case 0:
        range(a, b, ",");
        Bputc(&stdout, verb);
        range(c, d, ",");
        break;
    case 'n':
        Bprint(&stdout, "%s:", df->file1);
        range(a, b, ",");
        Bprint(&stdout, " %c ", verb);
        Bprint(&stdout, "%s:", df->file2);
        range(c, d, ",");
        break;
    case 'f':
        Bputc(&stdout, verb);
        range(a, b, " ");
        break;
    case 'c':
    case 'a':
    case 'u':
        if(df->nchanges%1024 == 0)
            df->changes = erealloc(df->changes, (df->nchanges+1024)*sizeof(df->changes[0]));
        ch = &df->changes[df->nchanges++];
        ch->oldx = a;
        ch->oldy = b;
        ch->newx = c;
        ch->newy = d;
    }
}

```

```

    return;
}
Bputc(&stdout, '\n');
if (mode == 0 || mode == 'n') {
    fetch(df, df->ixold, a, b, df->input[0], "< ");
    if (a <= b && c <= d)
        Bprint(&stdout, "---\n");
}
fetch(df, df->ixnew, c, d, df->input[1], mode == 0 || mode == 'n' ? "> ": "");
if (mode != 0 && mode != 'n' && c <= d)
    Bprint(&stdout, ".\n");
}

```

Uses `anychange 311b`, `fetch() 295b`, `mflag 311b`, `mode 311b`, and `stdout 311b`.

`<function changeset(diff) 297a>≡ (298)`

```

int
changeset(Diff *d, int i)
{
    while(i < d->nchanges && d->changes[i].oldy + 1 + 2*Lines > d->changes[i+1].oldx)
        i++;
    if(i < d->nchanges)
        return i+1;
    return d->nchanges;
}

```

Uses `Lines-7 298`.

`<function flushchanges(diff) 297b>≡ (298)`

```

void
flushchanges(Diff *df)
{
    vlong a, b, c, d, at, hdr;
    vlong i, j;

    if(df->nchanges == 0)
        return;

    hdr = 0;
    for(i=0; i < df->nchanges; ){
        j = changeset(df, i);
        a = df->changes[i].oldx - Lines;
        b = df->changes[j-1].oldy + Lines;
        c = df->changes[i].newx - Lines;
        d = df->changes[j-1].newy + Lines;
        if(a < 1)
            a = 1;
        if(c < 1)
            c = 1;
        if(b > df->len[0])
            b = df->len[0];
        if(d > df->len[1])
            d = df->len[1];
        if(mode == 'a'){
            a = 1;
            b = df->len[0];
            c = 1;
            d = df->len[1];
            j = df->nchanges;
        }
        if(mode == 'u'){
            if(!hdr){

```

```

        Bprint(&stdout, "--- %s\n", df->file1);
        Bprint(&stdout, "+++ %s\n", df->file2);
        hdr = 1;
    }
    Bprint(&stdout, "@@ -%lld,%lld +%lld,%lld @@\n", a, b-a+1, c, d-c+1);
}
else{
    Bprint(&stdout, "%s:", df->file1);
    range(a, b, ",");
    Bprint(&stdout, " - ");
    Bprint(&stdout, "%s:", df->file2);
    range(c, d, ",");
    Bputc(&stdout, '\n');
}
at = a;
for(; i<j; i++){
    fetch(df, df->ixold, at, df->changes[i].oldx-1, df->input[0], mode == 'u' ? " " : " ");
    fetch(df, df->ixold, df->changes[i].oldx, df->changes[i].oldy, df->input[0], mode == 'u' ? "-" : "-");
    fetch(df, df->ixnew, df->changes[i].newx, df->changes[i].newy, df->input[1], mode == 'u' ? "+" : "+");
    at = df->changes[i].oldy+1;
}
fetch(df, df->ixold, at, b, df->input[0], mode == 'u' ? " " : " ");
}
df->nchanges = 0;
}

```

Uses Lines-7 298, changeset() 297a, fetch() 295b, mode 311b, and stdout 311b.

<diff/diffio.c 298>≡

```

#include <u.h>
#include <libc.h>
#include <bio.h>
#include <ctype.h>
#include "diff.h"

```

<macro MIN(diff) 291b>

<function readline(diff) 292a>

<constant HALFLONG(diff) 292b>

<macro low(diff) 292c>

<macro high(diff) 292d>

<function readhash(diff) 292e>

<function prepare(diff) 293>

<function squishspace(diff) 294a>

<function check(diff) 294b>

<function range(diff) 295a>

<function fetch(diff) 295b>

<function change(diff) 296>

```

enum
{
    Lines = 3, /* number of lines of context shown */
};

```

<function changeset(diff) 297a>

<function flushchanges(diff) 297b>

D.2.5 diff/diffreg.c

<function sort(diff) 299a>≡ (305b)

```
static void
sort(Line *a, int n) /*shellsort CACM #201*/
{
    int m;
    Line *ai, *aim, *j, *k;
    Line w;
    int i;

    m = 0;
    for (i = 1; i <= n; i *= 2)
        m = 2*i - 1;
    for (m /= 2; m != 0; m /= 2) {
        k = a+(n-m);
        for (j = a+1; j <= k; j++) {
            ai = j;
            aim = ai+m;
            do {
                if (aim->value > ai->value ||
                    aim->value == ai->value &&
                    aim->serial > ai->serial)
                    break;
                w = *ai;
                *ai = *aim;
                *aim = w;

                aim = ai;
                ai -= m;
            } while (ai > a && aim >= ai);
        }
    }
}
```

<function unsort(diff) 299b>≡ (305b)

```
static void
unsort(Line *f, int l, int *b)
{
    int *a;
    int i;

    a = malloc((l+1)*sizeof(int));
    for(i=1;i<=l;i++)
        a[f[i].serial] = f[i].value;
    for(i=1;i<=l;i++)
        b[i] = a[i];
    free(a);
}
```

<function prune(diff) 299c>≡ (305b)

```
static void
prune(Diff *d)
{
    int i,j;
```

```

for(d->pref = 0; d->pref < d->len[0] && d->pref < d->len[1] &&
    d->file[0][d->pref+1].value == d->file[1][d->pref+1].value;
    d->pref++);
for(d->suff=0; d->suff < d->len[0] - d->pref && d->suff < d->len[1] - d->pref &&
    d->file[0][d->len[0] - d->suff].value == d->file[1][d->len[1] - d->suff].value;
    d->suff++);
for(j=0; j<2; j++) {
    d->sfile[j] = d->file[j]+d->pref;
    d->slen[j] = d->len[j]-d->pref-d->suff;
    for(i=0; i<=d->slen[j]; i++)
        d->sfile[j][i].serial = i;
}
}

```

<function equiv(diff) 300a> ≡ (305b)

```

static void
equiv(Line *a, int n, Line *b, int m, int *c)
{
    int i, j;

    i = j = 1;
    while(i<=n && j<=m) {
        if(a[i].value < b[j].value)
            a[i++].value = 0;
        else if(a[i].value == b[j].value)
            a[i++].value = j;
        else
            j++;
    }
    while(i <= n)
        a[i++].value = 0;
    b[m+1].value = 0;
    j = 0;
    while(++j <= m) {
        c[j] = -b[j].serial;
        while(b[j+1].value == b[j].value) {
            j++;
            c[j] = b[j].serial;
        }
    }
    c[j] = -1;
}

```

<function newcand(diff) 300b> ≡ (305b)

```

static int
newcand(Diff *d, int x, int y, int pred)
{
    Cand *q;

    d->clist = erealloc(d->clist, (d->cflen+1)*sizeof(Cand));
    q = d->clist + d->cflen;
    q->x = x;
    q->y = y;
    q->pred = pred;
    return d->cflen++;
}

```

<function search(diff) 300c> ≡ (305b)

```

static int

```

```

search(Diff *d, int *c, int k, int y)
{
    int i, j, l;
    int t;

    if(d->clist[c[k]].y < y)    /*quick look for typical case*/
        return k+1;
    i = 0;
    j = k+1;
    while((l=(i+j)/2) > i) {
        t = d->clist[c[l]].y;
        if(t > y)
            j = l;
        else if(t < y)
            i = l;
        else
            return l;
    }
    return l+1;
}

```

<function stone(diff) 301> ≡ (305b)

```

static int
stone(Diff *d, int *a, int n, int *b, int *c)
{
    int i, k,y;
    int j, l;
    int oldc, tc;
    int oldl;

    k = 0;
    c[0] = newcand(d, 0, 0, 0);
    for(i=1; i<=n; i++) {
        j = a[i];
        if(j==0)
            continue;
        y = -b[j];
        oldl = 0;
        oldc = c[0];
        do {
            if(y <= d->clist[oldc].y)
                continue;
            l = search(d, c, k, y);
            if(l!=oldl+1)
                oldc = c[l-1];
            if(l<=k) {
                if(d->clist[c[l]].y <= y)
                    continue;
                tc = c[l];
                c[l] = newcand(d, i, y, oldc);
                oldc = tc;
                oldl = l;
            } else {
                c[l] = newcand(d, i,y,oldc);
                k++;
                break;
            }
        } while((y=b[++j]) > 0);
    }
    return k;
}

```

```
}
```

Uses `newcand()` 300b and `search()` 300c.

<function unravel(diff) 302a>≡ (305b)

```
static void
unravel(Diff *d, int p)
{
    int i;
    Cand *q;

    for(i=0; i<=d->len[0]; i++) {
        if (i <= d->pref)
            d->J[i] = i;
        else if (i > d->len[0]-d->suff)
            d->J[i] = i+d->len[1] - d->len[0];
        else
            d->J[i] = 0;
    }
    for(q=d->clist+p; q->y != 0; q= d->clist + q->pred)
        d->J[q->x+d->pref] = q->y+d->pref;
}
```

<constant BUF(diff) 302b>≡ (305b)

```
#define BUF 4096
```

<function cmp(diff) 302c>≡ (305b)

```
static int
cmp(Biobuf* b1, Biobuf* b2)
{
    int n;
    uchar buf1[BUF], buf2[BUF];
    int f1, f2;
    vlong nc = 1;
    uchar *b1s, *b1e, *b2s, *b2e;

    f1 = Bfildes(b1);
    f2 = Bfildes(b2);
    seek(f1, 0, 0);
    seek(f2, 0, 0);
    b1s = b1e = buf1;
    b2s = b2e = buf2;
    for(;;){
        if(b1s >= b1e){
            if(b1s >= &buf1[BUF])
                b1s = buf1;
            n = read(f1, b1s, &buf1[BUF] - b1s);
            b1e = b1s + n;
        }
        if(b2s >= b2e){
            if(b2s >= &buf2[BUF])
                b2s = buf2;
            n = read(f2, b2s, &buf2[BUF] - b2s);
            b2e = b2s + n;
        }
        n = b2e - b2s;
        if(n > b1e - b1s)
            n = b1e - b1s;
        if(n <= 0)
            break;
        if(memcmp((void *)b1s, (void *)b2s, n) != 0){
```

```

        return 1;
    }
    nc += n;
    b1s += n;
    b2s += n;
}
if(b1e - b1s == b2e - b2s)
    return 0;
return 1;
}

```

Uses BUF-8 302b.

(function calcdiff(diff) 303)≡ (305b)

```

void
calcdiff(Diff *d, char *f, char *fo, char *t, char *to)
{
    Biobuf *b0, *b1;
    int k;

    b0 = prepare(d, 0, f, fo);
    if (!b0)
        return;
    b1 = prepare(d, 1, t, to);
    if (!b1) {
        Bterm(b0);
        return;
    }
    if (d->binary){
        // could use b0 and b1 but this is simpler.
        if(cmp(b0, b1))
            d->bindiff = 1;
        Bterm(b0);
        Bterm(b1);
        return;
    }
    d->cflen = 0;
    prune(d);
    sort(d->sfile[0], d->slen[0]);
    sort(d->sfile[1], d->slen[1]);

    d->member = (int *)d->file[1];
    equiv(d->sfile[0], d->slen[0], d->sfile[1], d->slen[1], d->member);
    d->member = erealloc(d->member, (d->slen[1]+2)*sizeof(int));

    d->class = (int *)d->file[0];
    unsort(d->sfile[0], d->slen[0], d->class);
    d->class = erealloc(d->class, (d->slen[0]+2)*sizeof(int));

    d->klist = emalloc((d->slen[0]+2)*sizeof(int));
    d->clist = emalloc(sizeof(Cand));
    k = stone(d, d->class, d->slen[0], d->member, d->klist);
    free(d->member);
    free(d->class);

    d->J = emalloc((d->len[0]+2)*sizeof(int));
    unravel(d, d->klist[k]);
    free(d->clist);
    free(d->klist);

    d->ixold = emalloc((d->len[0]+2)*sizeof(long));

```

```

d->ixnew = emalloc((d->len[1]+2)*sizeof(long));
Bseek(b0, 0, 0); Bseek(b1, 0, 0);
check(d, b0, b1);
}

```

Uses `check()` 294b, `cmp()` 302c, `equiv()` 300a, `prepare()` 293, `prune()` 299c, `sort()` 299a, `stone()` 301, `unravel()` 302a, and `unsort()` 299b.

`<function output(diff) 304a>≡ (305b)`

```

static void
output(Diff *d)
{
    int m, i0, i1, j0, j1;

    if(d->bindiff)
        print("binary files %s %s differ\n", d->file1, d->file2);
    if(d->binary)
        return;
    m = d->len[0];
    d->J[0] = 0;
    d->J[m+1] = d->len[1]+1;
    if (mode != 'e') {
        for (i0 = 1; i0 <= m; i0 = i1+1) {
            while (i0 <= m && d->J[i0] == d->J[i0-1]+1)
                i0++;
            j0 = d->J[i0-1]+1;
            i1 = i0-1;
            while (i1 < m && d->J[i1+1] == 0)
                i1++;
            j1 = d->J[i1+1]-1;
            d->J[i1] = j1;
            change(d, i0, i1, j0, j1);
        }
    } else {
        for (i0 = m; i0 >= 1; i0 = i1-1) {
            while (i0 >= 1 && d->J[i0] == d->J[i0+1]-1 && d->J[i0])
                i0--;
            j0 = d->J[i0+1]-1;
            i1 = i0+1;
            while (i1 > 1 && d->J[i1-1] == 0)
                i1--;
            j1 = d->J[i1-1]+1;
            d->J[i1] = j1;
            change(d, i1, i0, j1, j0);
        }
    }
    if (m == 0)
        change(d, 1, 0, 1, d->len[1]);
    flushchanges(d);
}

```

Uses `change()` 296, `flushchanges()` 297b, and `mode` 311b.

`<function diffreg(diff) 304b>≡ (305b)`

```

void
diffreg(char *f, char *fo, char *t, char *to)
{
    Diff d;

    memset(&d, 0, sizeof(d));
    calcdiff(&d, f, fo, t, to);
    output(&d);
}

```

```

    freediff(&d);
}

```

Uses `calcdiff()` 303, `freediff()` 305a, and `output()` 304a.

`<function freediff(diff) 305a>` ≡ (305b)

```

void
freediff(Diff *d)
{
    if(d->input[0] != nil)
        Bterm(d->input[0]);
    if(d->input[1] != nil)
        Bterm(d->input[1]);
    free(d->J);
    free(d->ixold);
    free(d->ixnew);
}

```

`<diff/diffreg.c 305b>` ≡

```

#include <u.h>
#include <libc.h>
#include <bio.h>

#include "diff.h"

/* diff - differential file comparison
 *
 * Uses an algorithm due to Harold Stone, which finds
 * a pair of longest identical subsequences in the two
 * files.
 *
 * The major goal is to generate the match vector J.
 * J[i] is the index of the line in file1 corresponding
 * to line i file0. J[i] = 0 if there is no
 * such line in file1.
 *
 * Lines are hashed so as to work in core. All potential
 * matches are located by sorting the lines of each file
 * on the hash (called value). In particular, this
 * collects the equivalence classes in file1 together.
 * Subroutine equiv replaces the value of each line in
 * file0 by the index of the first element of its
 * matching equivalence in (the reordered) file1.
 * To save space equiv squeezes file1 into a single
 * array member in which the equivalence classes
 * are simply concatenated, except that their first
 * members are flagged by changing sign.
 *
 * Next the indices that point into member are unsorted into
 * array class according to the original order of file0.
 *
 * The cleverness lies in routine stone. This marches
 * through the lines of file0, developing a vector klist
 * of "k-candidates". At step i a k-candidate is a matched
 * pair of lines x,y (x in file0 y in file1) such that
 * there is a common subsequence of length k
 * between the first i lines of file0 and the first y
 * lines of file1, but there is no such subsequence for
 * any smaller y. x is the earliest possible mate to y
 * that occurs in such a subsequence.
 *
 *

```

```

* Whenever any of the members of the equivalence class of
* lines in file1 matable to a line in file0 has serial number
* less than the y of some k-candidate, that k-candidate
* with the smallest such y is replaced. The new
* k-candidate is chained (via pred) to the current
* k-1 candidate so that the actual subsequence can
* be recovered. When a member has serial number greater
* than the y of all k-candidates, the klist is extended.
* At the end, the longest subsequence is pulled out
* and placed in the array J by unravel.
*
* With J in hand, the matches there recorded are
* check'ed against reality to assure that no spurious
* matches have crept in due to hashing. If they have,
* they are broken, and "jackpot " is recorded--a harmless
* matter except that a true match for a spuriously
* mated line may now be unnecessarily reported as a change.
*
* Much of the complexity of the program comes simply
* from trying to minimize core utilization and
* maximize the range of doable problems by dynamically
* allocating what is needed and reusing what is not.
* The core requirements for problems larger than somewhat
* are (in words) 2*length(file0) + length(file1) +
* 3*(number of k-candidates installed), typically about
* 6n words for files of length n.
*/

```

<function sort(diff) 299a>

<function unsort(diff) 299b>

<function prune(diff) 299c>

<function equiv(diff) 300a>

<function newcand(diff) 300b>

<function search(diff) 300c>

<function stone(diff) 301>

<function unravel(diff) 302a>

<constant BUF(diff) 302b>

<function cmp(diff) 302c>

<function calcdiff(diff) 303>

<function output(diff) 304a>

<function diffreg(diff) 304b>

<function freediff(diff) 305a>

D.2.6 diff/merge3.c

<macro min(diff) 306>≡

```
#define min(a, b) ((a)<(b)?(a):(b))
```

(310a)

```

<macro max(diff) 307a>≡ (310a)
#define max(a, b) ((a)>(b)?(a):(b))

```

```

<function changecmp(diff) 307b>≡ (310a)
static int
changecmp(void *a, void *b)
{
    return ((Change*)a)->oldx - ((Change*)b)->oldx;
}

```

```

<function addchange(diff) 307c>≡ (310a)
static void
addchange(Diff *df, int a, int b, int c, int d)
{
    Change *ch;

    if (a > b && c > d)
        return;
    if(df->nchanges%1024 == 0)
        df->changes = erealloc(df->changes, (df->nchanges+1024)*sizeof(df->changes[0]));
    ch = &df->changes[df->nchanges++];
    ch->oldx = a;
    ch->oldy = b;
    ch->newx = c;
    ch->newy = d;
}

```

```

<function collect(diff) 307d>≡ (310a)
static void
collect(Diff *d)
{
    int m, i0, i1, j0, j1;

    m = d->len[0];
    d->J[0] = 0;
    d->J[m+1] = d->len[1]+1;
    for (i0 = m; i0 >= 1; i0 = i1-1) {
        while (i0 >= 1 && d->J[i0] == d->J[i0+1]-1 && d->J[i0])
            i0--;
        j0 = d->J[i0+1]-1;
        i1 = i0+1;
        while (i1 > 1 && d->J[i1-1] == 0)
            i1--;
        j1 = d->J[i1-1]+1;
        d->J[i1] = j1;
        addchange(d, i1, i0, j1, j0);
    }
    if (m == 0)
        addchange(d, 1, 0, 1, d->len[1]);
    qsort(d->changes, d->nchanges, sizeof(Change), changecmp);
}

```

Uses `addchange()` 307c and `changecmp()` 307b.

```

<function overlaps(diff) 307e>≡ (310a)
static int
overlaps(int lx, int ly, int rx, int ry)
{
    if(lx <= rx)
        return ly >= ry;
    else

```

```

    return ry >= lx;
}

```

<function same(diff) 308a> ≡ (310a)

```

static int
same(Diff *l, Change *lc, Diff *r, Change *rc)
{
    char lbuf[MAXLINELEN], rbuf[MAXLINELEN];
    int i, ll, rl, lx, ly, rx, ry;

    lx = lc->newx;
    ly = lc->newy;
    rx = rc->newx;
    ry = rc->newy;
    if(ly - lx != ry - rx)
        return 0;
    assert(ly <= l->len[1] && ry <= r->len[1]);
    Bseek(l->input[1], l->ixnew[lx-1], 0);
    Bseek(r->input[1], r->ixnew[rx-1], 0);
    for(i = 0; i <= (ly - lx); i++){
        ll = readline(l->input[1], lbuf, sizeof(lbuf));
        rl = readline(r->input[1], rbuf, sizeof(rbuf));
        if(ll != rl)
            return 0;
        if(memcmp(lbuf, rbuf, ll) != 0)
            return 0;
    }
    return 1;
}

```

Uses MAXLINELEN 287c.

<function merge(diff) 308b> ≡ (310a)

```

char*
merge(Diff *l, Diff *r)
{
    int lx, ly, rx, ry;
    int il, ir, delta;
    Change *lc, *rc;
    char *status;
    vlong ln;

    il = 0;
    ir = 0;
    ln = 0;
    status = nil;
    collect(l);
    collect(r);
    while(il < l->nchanges || ir < r->nchanges){
        lc = nil;
        rc = nil;
        lx = -1;
        ly = -1;
        rx = -1;
        ry = -1;
        if(il < l->nchanges){
            lc = &l->changes[il];
            lx = (lc->oldx < lc->oldy) ? lc->oldx : lc->oldy;
            ly = (lc->oldx < lc->oldy) ? lc->oldy : lc->oldx;
        }
        if(ir < r->nchanges){

```

```

    rc = &r->changes[ir];
    rx = (rc->oldx < rc->oldy) ? rc->oldx : rc->oldy;
    ry = (rc->oldx < rc->oldy) ? rc->oldy : rc->oldx;
}
if(lc != nil && rc != nil && overlaps(lx, ly, rx, ry)){
    /*
     * align the edges of the chunks, expanding them
     * so that when we compare for sameness, we are
     * comparing same-sized chunks.
     */
    if(lc->oldx < rc->oldx){
        delta = rc->oldx - lc->oldx;
        rc->oldx = max(rc->oldx-delta, 1);
        rc->newx = max(rc->newx-delta, 1);
    }else{
        delta = lc->oldx - rc->oldx;
        lc->oldx = max(lc->oldx-delta, 1);
        lc->newx = max(lc->newx-delta, 1);
    }
    if(lc->oldy > rc->oldy){
        delta = lc->oldy - rc->oldy;
        rc->oldy = min(rc->oldy+delta, r->len[0]);
        rc->newy = min(rc->newy+delta, r->len[1]);
    }else{
        delta = rc->oldy - lc->oldy;
        lc->oldy = min(lc->oldy+delta, l->len[0]);
        lc->newy = min(lc->newy+delta, l->len[1]);
    }
    if(same(l, lc, r, rc)){
        fetch(l, l->ixold, ln, lc->oldx-1, l->input[0], "");
        fetch(l, l->ixnew, lc->newx, lc->newy, l->input[1], "");
    }else{
        fetch(l, l->ixold, ln, lc->oldx-1, l->input[0], "");
        Bprint(&stdout, "<<<<<<<<<< %s\n", l->file2);
        fetch(l, l->ixnew, lc->newx, lc->newy, l->input[1], "");
        Bprint(&stdout, "=====  
original\n");
        fetch(l, l->ixold, lc->oldx, lc->oldy, l->input[0], "");
        Bprint(&stdout, "=====  
%s\n", r->file2);
        fetch(r, r->ixnew, rc->newx, rc->newy, r->input[1], "");
        Bprint(&stdout, ">>>>>>>>>\n");
        status = "conflict";
    }
    ln = lc->oldy+1;
    il++;
    ir++;
}else if(lc != nil && (rc == nil || lx < rx)){
    fetch(l, l->ixold, ln, lc->oldx-1, l->input[0], "");
    fetch(l, l->ixnew, lc->newx, lc->newy, l->input[1], "");
    ln = lc->oldy+1;
    il++;
}else if(rc != nil && (lc == nil || rx < lx)){
    fetch(l, r->ixold, ln, rc->oldx-1, r->input[0], "");
    fetch(r, r->ixnew, rc->newx, rc->newy, r->input[1], "");
    ln = rc->oldy+1;
    ir++;
}else
    abort();
}
if(ln <= l->len[0])
    fetch(l, l->ixold, ln, l->len[0], l->input[0], "");

```

```

    return status;
}

```

Uses `collect()` 307d, `fetch()` 295b, `max-2` 307a, `min-1` 306, `overlaps()` 307e, `same()` 308a, and `stdout` 311b.

```

<diff/merge3.c 310a>≡
#include <u.h>
#include <libc.h>
#include <bio.h>

#include "diff.h"

<macro min(diff) 306>
<macro max(diff) 307a>

<function changecmp(diff) 307b>

<function addchange(diff) 307c>

<function collect(diff) 307d>

<function overlaps(diff) 307e>

<function same(diff) 308a>

<function merge(diff) 308b>

<function usage (diff/merge3.c)(diff) 229a>
<function main (diff/merge3.c)(diff) 229b>

```

D.2.7 diff/util.c

```

<function mkpathname(diff) 310b>≡ (311b)
int
mkpathname(char *pathname, char *path, char *name)
{
    if (strlen(path) + strlen(name) > MAXPATHLEN)
        sysfatal("pathname %s/%s too long", path, name);
    sprintf(pathname, "%s/%s", path, name);
    return 0;
}

```

Uses `MAXPATHLEN` 287b.

```

<function mktmpfile(diff) 310c>≡ (311b)
char *
mktmpfile(int input, Dir **sb)
{
    int fd, i;
    char *p;
    char buf[8192];

    p = mktemp(tmp[whichtmp++]);
    /*
     * Because we want this file to stick around
     * for the entire run of the program, we leak
     * the fd intentionally here; when we exit,
     * the system will remove the file for us.
     */
    fd = create(p, OWRITE|ORCLOSE, 0600);
}

```

```

if (fd < 0)
    sysfatal("cannot create %s: %r", p);
while ((i = read(input, buf, sizeof(buf))) > 0) {
    if ((i = write(fd, buf, i)) < 0)
        break;
}
*sb = dirfstat(fd);
if (i < 0)
    sysfatal("cannot read/write %s: %r", p);
return p;
}

```

Uses tmp-9 311b and whichtmp-10 311b.

```

<function statfile(diff) 311a>≡ (311b)
char *
statfile(char *file, Dir **sb)
{
    Dir *dir;
    int input;

    dir = dirstat(file);
    if(dir == nil) {
        if (strcmp(file, "-") || (dir = dirfstat(0)) == nil)
            sysfatal("cannot stat %s: %r", file);
        free(dir);
        return mktmpfile(0, sb);
    } else if (!REGULAR_FILE(dir) && !DIRECTORY(dir)) {
        free(dir);
        if ((input = open(file, OREAD)) == -1)
            sysfatal("cannot open %s: %r", file);
        file = mktmpfile(input, sb);
        close(input);
    } else
        *sb = dir;
    return file;
}

```

Uses DIRECTORY 287d, REGULAR_FILE 287e, and mktmpfile() 310c.

```

<diff/util.c 311b>≡
#include <u.h>
#include <libc.h>
#include <bio.h>
#include "diff.h"

Biobuf stdout;
char mode; /* '\0', 'e', 'f', 'h' */
char bflag; /* ignore multiple and trailing blanks */
char rflag; /* recurse down directory trees */
char mflag; /* pseudo flag: doing multiple files, one dir */
int anychange;

static char *tmp[] = {"/tmp/diff1XXXXXXXXXXXX", "/tmp/diff2XXXXXXXXXXXX"};
static int whichtmp;

<function emalloc(diff) 257c>
<function erealloc(diff) 257d>

<function mkpathname(diff) 310b>
<function mktmpfile(diff) 310c>

```

<function statfile(diff) 311a>

D.3 misc/

D.3.1 misc/patch.c

```
<function readline(patch.c) 312a>≡ (323b)
char*
readline(Biobuf *f, int *lnum)
{
    char *ln;

    if((ln = Brdstr(f, '\n', 0)) == nil)
        return nil;
    *lnum += 1;
    return ln;
}
```

```
<function fail(patch.c) 312b>≡ (323b)
void
fail(char *fmt, ...)
{
    char msg[ERRMAX];
    va_list ap;

    finish(0);
    va_start(ap, fmt);
    vsnprint(msg, sizeof(msg), fmt, ap);
    va_end(ap);
    fprintf(2, "%s\n", msg);
    exits(msg);
}
```

Uses `finish()` 319.

```
<function fileheader(patch.c) 312c>≡ (323b)
int
fileheader(char *s, char *pfx, char **name)
{
    int len, n, nnull;
    char *e;

    if((strncmp(s, pfx, strlen(pfx))) != 0)
        return -1;
    for(s += strlen(pfx); *s; s++)
        if(!isspace(*s))
            break;
    for(e = s; *e; e++)
        if(isspace(*e))
            break;
    if(s == e)
        return -1;
    nnull = strlen("/dev/null");
    if((e - s) != nnull || strncmp(s, "/dev/null", nnull) != 0){
        n = strip;
        while(s != e && n > 0){
            while(s != e && *s == '/')
                s++;
            while(s != e && *s != '/')

```

```

        s++;
        n--;
    }
    while(*s == '/')
        s++;
    if(*s == '\0')
        fail("too many components stripped");
}
len = (e - s) + 1;
*name = emalloc(len);
strecpy(*name, *name + len, s);
return 0;
}

```

<function hunkheader(patch.c) 313> ≡ (323b)

```

int
hunkheader(Hunk *h, char *s, char *oldpath, char *newpath, int lnum)
{
    char *e;

    memset(h, 0, sizeof(*h));
    h->lnum = lnum;
    h->oldpath = strdup(oldpath);
    h->newpath = strdup(newpath);
    h->origlen = 0;
    h->origsz = 32;
    h->orig = emalloc(h->origsz);
    h->oldlen = 0;
    h->oldsz = 32;
    h->old = emalloc(h->oldsz);
    h->newlen = 0;
    h->newsz = 32;
    h->new = emalloc(h->newsz);
    cleannname(h->oldpath);
    cleannname(h->newpath);
    if(strncmp(s, "@@ -", 4) != 0)
        return -1;
    e = s + 4;
    h->oldln = strtol(e, &e, 10);
    h->oldcnt = 1;
    if(*e == ','){
        e++;
        h->oldcnt = strtol(e, &e, 10);
    }
    while(*e == ' ' || *e == '\t')
        e++;
    if(*e != '+')
        return -1;
    e++;
    h->newln = strtol(e, &e, 10);
    h->newcnt = 1;
    if(e == s)
        return -1;
    if(*e == ','){
        e++;
        h->newcnt = strtol(e, &e, 10);
    }
    if(e == s || *e != ' ')
        return -1;
    if(strncmp(e, "@@", 3) != 0)

```

```

    return -1;
/*
 * empty files have line number 0: keep that,
 * otherwise adjust down.
 */
if(h->oldln > 0)
    h->oldln--;
if(h->newln > 0)
    h->newln--;
if(h->oldln < 0 || h->newln < 0 || h->oldcnt < 0 || h->newcnt < 0)
    fail("malformed hunk %s", s);
return 0;
}

```

<function addorig(patch.c) 314a> ≡ (323b)

```

void
addorig(Hunk *h, char *ln)
{
    int n;

    n = strlen(ln);
    while(h->origlen + n >= h->origsz){
        h->origsz *= 2;
        h->orig = erealloc(h->orig, h->origsz);
    }
    memcpy(h->orig + h->origlen, ln, n);
    h->origlen += n;
}

```

<function addnew(patch.c) 314b> ≡ (323b)

```

void
addnew(Hunk *h, char *ln)
{
    int n;

    ln++;
    n = strlen(ln);
    while(h->newlen + n >= h->newsz){
        h->newsz *= 2;
        h->new = erealloc(h->new, h->newsz);
    }
    memcpy(h->new + h->newlen, ln, n);
    h->newlen += n;
}

```

<function addold(patch.c) 314c> ≡ (323b)

```

void
addold(Hunk *h, char *ln)
{
    int n;

    ln++;
    n = strlen(ln);
    while(h->oldlen + n >= h->oldsz){
        h->oldsz *= 2;
        h->old = erealloc(h->old, h->oldsz);
    }
    memcpy(h->old + h->oldlen, ln, n);
    h->oldlen += n;
}

```

```

<function addmiss(patch.c) 315a>≡ (323b)
int
addmiss(Hunk *h, char *ln, int *nold, int *nnew)
{
    if(ln == nil)
        return 1;
    else if(ln[0] != '-' && ln[0] != '+')
        return 0;
    if(ln[0] == '-'){
        addold(h, ln);
        *nold += 1;
    }else{
        addnew(h, ln);
        *nnew += 1;
    }
    return 1;
}

```

Uses addnew() 314b and addold() 314c.

```

<function addhunk(patch.c) 315b>≡ (323b)
void
addhunk(Patch *p, Hunk *h)
{
    p->hunk = erealloc(p->hunk, ++p->nhunk*sizeof(Hunk));
    p->hunk[p->nhunk-1] = *h;
}

```

```

<function hunkcmp(patch.c) 315c>≡ (323b)
int
hunkcmp(void *a, void *b)
{
    int c;

    c = strcmp(((Hunk*)a)->oldpath, ((Hunk*)b)->oldpath);
    if(c != 0)
        return c;
    return ((Hunk*)a)->oldln - ((Hunk*)b)->oldln;
}

```

```

<function swapint(patch.c) 315d>≡ (323b)
void
swapint(int *a, int *b)
{
    int t;

    t = *a;
    *a = *b;
    *b = t;
}

```

```

<function swapstr(patch.c) 315e>≡ (323b)
void
swapstr(char **a, char **b)
{
    char *t;

    t = *a;
    *a = *b;
    *b = t;
}

```

```

⟨function trimhunk(patch.c) 316a⟩≡ (323b)
void
trimhunk(char c, Hunk *h)
{
    if((c == ' ' || c == '-') && h->oldlen > 0 && h->old[h->oldlen-1] == '\n'){
        h->oldcnt--;
        h->oldlen--;
    }
    if((c == ' ' || c == '+') && h->newlen > 0 && h->new[h->newlen-1] == '\n'){
        h->newcnt--;
        h->newlen--;
    }
}

```

```

⟨function parse(patch.c) 316b⟩≡ (323b)
Patch*
parse(Biobuf *f, char *name)
{
    char *ln, *old, *new, c;
    int i, oldcnt, newcnt, lnum;
    Patch *p;
    Hunk h, *ph;

    ln = nil;
    lnum = 0;
    p = emalloc(sizeof(Patch));
comment:
    free(ln);
    while((ln = readline(f, &lnum)) != nil){
        if(strncmp(ln, "--- ", 4) == 0)
            goto patch;
        free(ln);
    }
    if(p->nhunk == 0)
        fail("%s: could not find start of patch", name);
    goto out;

patch:
    if(fileheader(ln, "--- ", &old) == -1)
        goto comment;
    free(ln);

    if((ln = readline(f, &lnum)) == nil)
        goto out;
    if(fileheader(ln, "+++ ", &new) == -1)
        goto comment;
    free(ln);

    if((ln = readline(f, &lnum)) == nil)
        goto out;
hunk:
    oldcnt = 0;
    newcnt = 0;
    if(hunkheader(&h, ln, old, new, lnum) == -1)
        goto comment;
    free(ln);

    while(1){
        if((ln = readline(f, &lnum)) == nil){
            if(oldcnt != h.oldcnt)

```

```

        fail("%s:%d: malformed hunk: mismatched -hunk size %d != %d", name, lnum, oldcnt, h.oldcnt);
    if(newcnt != h.newcnt)
        fail("%s:%d: malformed hunk: mismatched +hunk size %d != %d", name, lnum, newcnt, h.newcnt);
    addhunk(p, &h);
    break;
}
c = ln[0];
addorig(&h, ln);
switch(ln[0]){
default:
    fail("%s:%d: malformed hunk: leading junk", name, lnum);
case '\\':
    if(strncmp(ln, "\\ No newline", nelem("\\ No newline")-1) == 0)
        trimhunk(c, &h);
    /* ignore unknown directives */
    break;
case '-':
    addold(&h, ln);
    oldcnt++;
    break;
case '+':
    addnew(&h, ln);
    newcnt++;
    break;
case '\\n':
    addold(&h, " \n");
    addnew(&h, " \n");
    oldcnt++;
    newcnt++;
    break;
case ' ':
    addold(&h, ln);
    addnew(&h, ln);
    oldcnt++;
    newcnt++;
    break;
}
free(ln);
if(oldcnt > h.oldcnt || newcnt > h.newcnt)
    fail("%s:%d: malformed hunk: oversized hunk", name, lnum);
if(oldcnt < h.oldcnt || newcnt < h.newcnt)
    continue;

addhunk(p, &h);
if((ln = readline(f, &lnum)) == nil)
    goto out;
if(strncmp(ln, "\\ No newline", nelem("\\ No newline")-1) == 0)
    trimhunk(c, &p->hunk[p->nhunk-1]);
if(strncmp(ln, "--- ", 4) == 0)
    goto patch;
if(strncmp(ln, "@@ ", 3) == 0)
    goto hunk;
goto comment;
}

out:
if(reverse){
    for(i = 0; i < p->nhunk; i++){
        ph = &p->hunk[i];
        swapint(&ph->oldln, &ph->newln);
    }
}

```

```

        swapint(&ph->oldcnt, &ph->newcnt);
        swapint(&ph->oldlen, &ph->newlen);
        swapint(&ph->oldsz, &ph->newsz);
        swapstr(&ph->oldpath, &ph->newpath);
        swapstr(&ph->old, &ph->new);
    }
}
qsort(p->hunk, p->nhunk, sizeof(Hunk), hunkcmp);
free(old);
free(new);
free(ln);
return p;
}

```

Uses `addhunk()` 315b, `addnew()` 314b, `addold()` 314c, `addorig()` 314a, `hunkcmp()` 315c, `hunkheader()` 313, `swapint()` 315d, `swapstr()` 315e, and `trimhunk()` 316a.

<function rename(patch.c) 318a> ≡ (323b)

```

int
rename(int fd, char *name)
{
    Dir st;
    char *p;

    nulldir(&st);
    if((p = strrchr(name, '/')) == nil)
        st.name = name;
    else
        st.name = p + 1;
    return dirfstat(fd, &st);
}

```

<function mkpath(patch.c) 318b> ≡ (323b)

```

int
mkpath(char *path)
{
    char *p, buf[ERRMAX];
    int f;

    if(*path == '\0')
        return 0;
    for(p = strchr(path+1, '/'); p != nil; p = strchr(p+1, '/')){
        *p = '\0';
        if(access(path, AEXIST) != 0){
            if((f = create(path, OREAD, DMDIR | 0777)) == -1){
                rerrstr(buf, sizeof(buf));
                if(strstr(buf, "exist") == nil)
                    return -1;
            }
            close(f);
        }
        *p = '/';
    }
    return 0;
}

```

<function blat(patch.c) 318c> ≡ (323b)

```

void
blat(char *old, char *new, char *o, usize len, int mode)
{
    char *tmp;

```

```

int fd;

tmp = nil;
if(strcmp(old, "/dev/null") == 0 && !dryrun)
    if(access(new, AEXIST) != -1)
        fail("would clobber: %s", new);
if(strcmp(new, "/dev/null") != 0 && !dryrun){
    if(mkpath(new) == -1)
        fail("mkpath %s: %r", new);
    if((tmp = smprint("%s.tmp%d", new, getpid())) == nil)
        fail("smprint: %r");
    if((fd = create(tmp, OWRITE, mode|0200)) == -1)
        fail("open %s: %r", tmp);
    if(write(fd, o, len) != len)
        fail("write %s: %r", tmp);
    close(fd);
}
if((changed = realloc(changed, (nchanged+1)*sizeof(Fchg))) == nil)
    fail("realloc: %r");
if((changed[nchanged].new = strdup(new)) == nil)
    fail("strdup: %r");
if((changed[nchanged].old = strdup(old)) == nil)
    fail("strdup: %r");
changed[nchanged].tmp = tmp;
nchanged++;
}

```

Uses `changed` [323b](#), `dryrun` [323b](#), and `nchanged` [323b](#).

<function finish(patch.c) 319> ≡ (323b)

```

int
finish(int ok)
{
    Fchg *c;
    int i, fd;

    for(i = 0; i < nchanged; i++){
        c = &changed[i];
        if(!ok){
            if(c->tmp != nil && remove(c->tmp) == -1)
                fprintf(2, "remove %s: %r\n", c->tmp);
            goto Free;
        }
        if(!dryrun){
            if(strcmp(c->new, "/dev/null") == 0){
                if(remove(c->old) == -1){
                    ok = 0;
                    fprintf(2, "remove %s: %r\n", c->old);
                    goto Free;
                }
                goto Print;
            }
            if((fd = open(c->tmp, ORDWR)) == -1){
                ok = 0;
                fprintf(2, "open %s: %r\n", c->tmp);
                goto Free;
            }
            if(strcmp(c->old, c->new) == 0 && remove(c->old) == -1
                || rename(fd, c->new) == -1
                || close(fd) == -1){
                ok = 0;
            }
        }
    }
}

```

```

        fprintf(2, "cleanup: %r");
    }
}
Print:
    if(strcmp(c->new, "/dev/null") == 0)
        print("%s\n", c->old);
    else
        print("%s\n", c->new);
Free:
    free(c->tmp);
    free(c->old);
    free(c->new);
}
free(changed);
return ok;
}

```

Uses `changed` 323b, `dryrun` 323b, and `nchanged` 323b.

```

⟨function slurp(patch.c) 320⟩≡ (323b)
int
slurp(Fbuf *f, char *path)
{
    int n, i, fd, sz, len, nlines, linesz;
    char *buf;
    int *lines;
    Dir *d;

    if((fd = open(path, OREAD)) == -1){
        fprintf(2, "open %s: %r", path);
        return -1;
    }
    if((d = dirfstat(fd)) == nil)
        fail("stat %s: %r", path);
    sz = 8192;
    len = 0;
    buf = emalloc(sz);
    while(1){
        if(len == sz){
            sz *= 2;
            buf = erealloc(buf, sz);
        }
        n = read(fd, buf + len, sz - len);
        if(n == 0)
            break;
        if(n == -1)
            fail("read %s: %r", path);
        len += n;
    }

    nlines = 0;
    linesz = 32;
    lines = emalloc(linesz*sizeof(int));
    lines[nlines++] = 0;
    for(i = 0; i < len; i++){
        if(buf[i] != '\n')
            continue;
        if(nlines+2 == linesz){
            linesz *= 2;
            lines = erealloc(lines, linesz*sizeof(int));
        }
    }
}

```

```

        lines[nlines++] = i+1;
    }
    lines[nlines] = len;
    f->len = len;
    f->buf = buf;
    f->lines = lines;
    f->nlines = nlines;
    f->lastln = 0;
    f->lastfuzz = 0;
    f->mode = d->mode;
    free(d);
    close(fd);
    return 0;
}

```

<function searchln(patch.c) 321a>≡ (323b)

```

char*
searchln(Fbuf *f, Hunk *h, int ln)
{
    int off;

    off = f->lines[ln];
    if(off + h->oldlen > f->len)
        return nil;
    if(memcmp(f->buf + off, h->old, h->oldlen) != 0)
        return nil;
    f->lastln = ln + h->oldcnt;
    f->lastfuzz = ln - h->oldln;
    return f->buf + off;
}

```

<function search(patch.c) 321b>≡ (323b)

```

char*
search(Fbuf *f, Hunk *h)
{
    int ln, oldln, fuzz, scanning;
    char *p;

    oldln = h->oldln + f->lastfuzz;
    if(oldln + h->oldcnt > f->nlines)
        oldln = f->nlines - h->oldcnt;
    scanning = oldln >= f->lastln;
    for(fuzz = 0; scanning && fuzz < 250; fuzz++){
        scanning = 0;
        ln = oldln - fuzz;
        if(ln >= f->lastln){
            scanning = 1;
            if((p = searchln(f, h, ln)) != nil)
                return p;
        }
        ln = oldln + fuzz + 1;
        if(ln + h->oldcnt <= f->nlines){
            scanning = 1;
            if((p = searchln(f, h, ln)) != nil)
                return p;
        }
    }
    return nil;
}

```

```

⟨function rejected(patch.c) 322a⟩≡ (323b)
void
rejected(Hunk *h, char *fname)
{
    fprintf(2, "%s:%d: skipping failed hunk %s:%d\n", fname, h->lnum, h->oldpath, h->oldln);
    fprintf(rejfd, "--- %s:%d \n", h->oldpath, h->oldln);
    fprintf(rejfd, "+++ %s:%d \n", h->newpath, h->newln);
    fprintf(rejfd, "@@ -%d,%d +%d,%d @@\n", h->oldln, h->oldcnt, h->newln, h->newcnt);
    write(rejfd, h->orig, h->origlen);
}

```

Uses rejfd 323b.

```

⟨function append(patch.c) 322b⟩≡ (323b)
char*
append(char *o, int *sz, char *s, char *e)
{
    int n;

    n = (e - s);
    o = erealloc(o, *sz + n);
    memcpy(o + *sz, s, n);
    *sz += n;
    return o;
}

```

```

⟨function apply(patch.c) 322c⟩≡ (323b)
int
apply(Patch *p, char *fname)
{
    char *o, *s, *n, *curfile, *nextfile;
    int i, osz;
    Hunk *h, *prevh;
    Fbuf f;

    n = nil;
    o = nil;
    osz = 0;
    curfile = nil;
    h = nil;
    prevh = nil;
    nchanged = 0;
    changed = nil;
    for(i = 0; i < p->nhunk; i++){
        h = &p->hunk[i];
        if(strcmp(h->newpath, "/dev/null") == 0)
            nextfile = h->oldpath;
        else
            nextfile = h->newpath;
        if(curfile == nil || strcmp(curfile, nextfile) != 0){
            if(curfile != nil){
                if(!dryrun)
                    o = append(o, &osz, n, f.buf + f.len);
                blat(prevh->oldpath, prevh->newpath, o, osz, f.mode);
                osz = 0;
            }
            if(!dryrun){
                if(slurp(&f, h->oldpath) == -1){
                    rejected(h, fname);
                    goto Next;
                }
            }
        }
    }
}

```

```

        n = f.buf;
    }
    curfile = nextfile;
}
if(!dryrun){
    char *e;
    s = n;
    e = search(&f, h);
    if(e == nil){
        if(rejfd == -1)
            fail("%s:%d: unable to find hunk offset near %s:%d", fname, h->lnum, h->oldpath, h->oldln);
        else{
            rejected(h, fname);
            goto Next;
        }
    }
    o = append(o, &osz, s, e);
    o = append(o, &osz, h->new, h->new + h->newlen);
    n = e + h->oldlen;
}
Next:
    prevh = h;
}
if(curfile != nil){
    if(!dryrun)
        o = append(o, &osz, n, f.buf + f.len);
    blat(h->oldpath, h->newpath, o, osz, f.mode);
}
free(o);
return 0;
}

```

Uses `blat()` 318c, `changed` 323b, `dryrun` 323b, `nchanged` 323b, `rejected()` 322a, `rejfd` 323b, `search()` 300c, and `slurp()` 320.

`<function freepatch(patch.c) 323a>≡ (323b)`

```

void
freepatch(Patch *p)
{
    Hunk *h;
    int i;

    for(i = 0; i < p->nhunk; i++){
        h = &p->hunk[i];
        free(h->oldpath);
        free(h->newpath);
        free(h->old);
        free(h->new);
    }
    free(p->hunk);
    free(p->name);
    free(p);
}

```

`<misc/patch.c 323b>≡`

```

#include <u.h>
#include <libc.h>
#include <ctype.h>
#include <bio.h>

```

```

typedef struct Patch Patch;
typedef struct Hunk Hunk;

```

```

typedef struct Fbuf Fbuf;
typedef struct Fchg Fchg;

struct Patch {
    char    *name;
    Hunk    *hunk;
    usize   nhunk;
};

struct Hunk {
    int lnum;

    char    *orig;
    int origlen;
    int origsz;

    char    *oldpath;
    int oldln;
    int oldcnt;
    int oldlen;
    int oldsz;
    char    *old;

    char    *newpath;
    int newln;
    int newcnt;
    int newlen;
    int newsz;
    char    *new;
};

struct Fbuf {
    int *lines;
    int nlines;
    int lastln;
    int lastfuzz;
    char *buf;
    int len;
    int mode;
};

struct Fchg {
    char *tmp;
    char *old;
    char *new;
};

int finish(int);
void fail(char*, ...);

int strip;
int reverse;
int rejfd = -1;
char *rejfile;
char *workdir;
Fchg *changed;
int nchanged;
int dryrun;

```

<function readline(patch.c) 312a>

<function emalloc(patch.c) 257e>
<function erealloc(patch.c) 258a>
<function fail(patch.c) 312b>
<function fileheader(patch.c) 312c>
<function hunkheader(patch.c) 313>
<function addorig(patch.c) 314a>
<function addnew(patch.c) 314b>
<function addold(patch.c) 314c>
<function addmiss(patch.c) 315a>
<function addhunk(patch.c) 315b>
<function hunkcmp(patch.c) 315c>
<function swapint(patch.c) 315d>
<function swapstr(patch.c) 315e>
<function trimhunk(patch.c) 316a>
<function parse(patch.c) 316b>
<function rename(patch.c) 318a>
<function mkpath(patch.c) 318b>
<function blat(patch.c) 318c>
<function finish(patch.c) 319>
<function slurp(patch.c) 320>
<function searchln(patch.c) 321a>
<function search(patch.c) 321b>
<function rejected(patch.c) 322a>
<function append(patch.c) 322b>
<function apply(patch.c) 322c>
<function freepatch(patch.c) 323a>
<function usage(patch.c) 227b>
<function main(patch.c) 228>

Uses Fbuf 323b, Fchg 323b, Hunk 323b, Patch 323b, and rejfd 323b.

Glossary

VCS = Version Control System

DVCS = Distributed Version Control System

SHA1 = Secure Hash Algorithm 1

URL = Uniform Resource Locator

Indexes

Here is a list of the identifiers used, and where they appear. Underlined entries indicate the place of definition. This index is generated automatically.

Accepthdr-32: [245b](#), [246e](#)
addblk(): [154e](#), [155b](#), [155b](#)
addchange(): [307c](#), [307d](#)
addhunk(): [315b](#), [316b](#)
addmeta(): [152e](#), [153b](#), [173b](#), [173b](#)
admiss(): [315a](#)
addnew(): [314b](#), [315a](#), [316b](#)
addold(): [314c](#), [315a](#), [316b](#)
addorig(): [314a](#), [316b](#)
Aflg-14: [40b](#)
allowwrite: [215b](#), [222a](#)
ancestor(): [94a](#), [199](#)
anychange: [296](#), [311b](#)
append(): [166](#)
apply(): [322c](#)
applydelta(): [46e](#), [178](#), [180e](#)
authoremail: [58c](#), [118](#)
authorname: [118](#)
authorpat: [49a](#), [259b](#), [284b](#)
bappend(): [160e](#), [276b](#)
bdecompress(): [177d](#), [276b](#), [276b](#)
bdir: [109e](#), [109e](#), [109f](#), [110](#)
bflag: [292e](#), [294a](#), [294b](#), [311b](#)
Blank-17: [95e](#)
blat(): [318c](#), [322c](#)
blobify(): [57c](#)
branch: [201b](#), [202e](#)
branches: [66d](#), [66d](#), [85d](#)
branchgen(): [85b](#), [85c](#)
branchid(): [85c](#), [85d](#), [86a](#)
branchmatch(): [186c](#), [194a](#)
breadc(): [276b](#), [276b](#)
Buf: [270](#), [270](#)
BUF-8: [302b](#), [302c](#)
Buf (typedef): [270](#)
bwrite(): [282b](#)
Cache: [267](#)
cache(): [32c](#), [34a](#), [35c](#)

Cache.cache: [69a](#)
Cache.max: [69a](#)
Cache.n: [69a](#)
Cache (typedef): [267](#)
cachemax: [249d](#)
calcdiff(): [303](#), [304b](#)
Cand: [287f](#), [287f](#)
Cand.pred: [287f](#)
Cand.x: [287f](#)
Cand.y: [287f](#)
Cand (typedef): [287f](#)
Ccache: [33c](#), [33e](#), [249d](#), [277](#)
Cexist: [158a](#), [277](#)
Change: [287f](#), [287f](#)
change(): [296](#), [304a](#)
Change.newx: [287f](#)
Change.newy: [287f](#)
Change.oldx: [287f](#)
Change.oldy: [287f](#)
Change (typedef): [287f](#)
changecmp(): [307b](#), [307d](#)
changed: [318c](#), [319](#), [322c](#), [323b](#)
changes: [87b](#), [98b](#)
changeset(): [297a](#), [297b](#)
charval(): [31d](#), [51f](#)
chattygit: [199](#), [253c](#), [254f](#)
check(): [294b](#), [303](#)
checkedin(): [109f](#)
checkhash(): [218b](#)
Cidx: [33c](#), [47c](#), [236b](#), [277](#)
Cinfo: [270](#), [270](#)
Cinfo (typedef): [270](#)
cleanidx: [109f](#), [285](#), [285](#)
cleanup(): [151](#)
clear(): [35a](#), [160c](#), [180e](#)
clearedobject(): [43b](#), [153b](#)
Cloaded: [34a](#), [35c](#), [47c](#), [161e](#), [177c](#), [237b](#)
closeconn(): [186c](#), [208d](#), [245c](#)
closepack(): [45d](#), [158d](#)
cmp(): [302c](#), [303](#)
collect(): [307d](#), [308b](#)
commitid: [272](#)
commitmsg: [118](#), [282b](#)
committeremail: [118](#), [282b](#)
committername: [118](#)
Compout: [277](#)
Compout.bfd: [154a](#)
Compout.st: [154a](#)
Compout (typedef): [277](#)

compread(): [163c](#), [277](#)
compwrite(): [163c](#), [277](#)
Conn: [270](#), [270](#)
Conn (typedef): [270](#)
ConnGit: [186a](#), [207b](#), [245c](#), [247b](#), [249a](#)
ConnGit9: [245c](#), [247b](#)
ConnHttp: [206b](#), [242g](#), [245c](#)
ConnSsh: [33c](#), [245c](#), [246b](#)
Contenthdr-31: [245a](#), [246a](#)
Cparsed: [34a](#), [35c](#), [47c](#), [48c](#)
crackidx(): [273b](#), [274b](#)
crc32(): [52a](#)
Crumb: [67c](#), [267](#)
crumb(): [67d](#), [71b](#), [72a](#), [72c](#), [73](#), [74a](#), [74b](#), [74d](#), [76d](#), [78b](#), [79b](#), [80c](#), [85c](#), [236f](#)
Crumb.mode: [67c](#)
Crumb.mtime: [67c](#)
Crumb.name: [67c](#)
Crumb.obj: [67c](#)
Crumb.qid: [67c](#)
Crumb (typedef): [267](#)
Cthin: [178](#), [180a](#), [236b](#)
Cxxx: [33a](#)
Dblock: [270](#), [270](#)
Dblock (typedef): [270](#)
decompress(): [161e](#), [178](#), [276b](#)
Delta: [270](#), [270](#)
Delta (typedef): [270](#)
deltaordercmp(): [165b](#), [173b](#)
deltasz(): [164d](#), [165b](#)
deltify(): [164d](#), [167b](#)
dialgit(): [212b](#), [279](#)
dialhgit(): [185b](#), [208d](#)
dialhttp(): [185b](#)
dialssh(): [185d](#), [242g](#)
Diff: [287f](#), [287f](#)
diff(): [289c](#), [290](#)
Diff.b0: [287f](#)
Diff.b1: [287f](#)
Diff.binary: [287f](#)
Diff.bindiff: [287f](#)
Diff.cand: [287f](#)
Diff.changes: [287f](#)
Diff.class: [287f](#)
Diff.clen: [287f](#)
Diff.clist: [287f](#)
Diff.file: [287f](#)
Diff.file1: [287f](#)
Diff.file2: [287f](#)
Diff.firstchange: [287f](#)

Diff.input: [287f](#)
Diff.ixnew: [287f](#)
Diff.ixold: [287f](#)
Diff.J: [287f](#)
Diff.klist: [287f](#)
Diff.len: [287f](#)
Diff.line: [287f](#)
Diff.member: [287f](#)
Diff.nchanges: [287f](#)
Diff.pref: [287f](#)
Diff.sfile: [287f](#)
Diff.slen: [287f](#)
Diff.suff: [287f](#)
Diff (typedef): [287f](#)
diffcommits(): [98e](#)
diffdir(): [289c](#), [290](#)
diffreg(): [290](#), [304b](#)
difftrees(): [98e](#), [99](#), [99](#)
DIRECTORY: [287d](#), [290](#), [311a](#)
Dirent: [270](#), [270](#)
Dirent (typedef): [270](#)
dirty: [103b](#)
done: [146a](#), [227a](#)
done(): [146a](#), [227a](#)
dprint: [165b](#), [172c](#), [201b](#), [206b](#), [207b](#), [209a](#), [209c](#), [213a](#), [217b](#), [243b](#), [245c](#), [246b](#), [247b](#), [270](#)
Drop-19: [95e](#)
dryrun: [318c](#), [319](#), [322c](#), [323b](#)
Dtab: [270](#), [270](#)
Dtab (typedef): [270](#)
dtclear(): [154d](#), [163c](#), [165b](#)
dtinit(): [154e](#), [165b](#)
eamalloc(): [37c](#), [38b](#), [39f](#), [49b](#), [70c](#), [89a](#), [154e](#), [155b](#), [160b](#), [186c](#), [199](#), [201b](#), [250](#)
earealloc(): [40a](#), [49b](#), [169a](#), [173b](#), [186c](#), [199](#), [257a](#)
eatspace(): [90b](#)
Ebadobj-67: [77b](#), [79e](#)
Eexist-60: [78a](#), [79e](#), [81b](#), [86a](#)
emalloc(): [257c](#)
emitdelta(): [167b](#), [169a](#)
emptydir(): [93a](#), [118](#), [237d](#)
encodedelta(): [166](#), [172c](#)
endswith(): [259c](#), [274b](#)
Enodir-63: [82f](#)
enqueueparent(): [186c](#), [194d](#)
entcmp(): [35c](#), [99](#)
Eperm-59: [71b](#)
equiv(): [300a](#), [303](#)
erealloc(): [257d](#)
estrdup(): [56d](#), [70b](#), [70c](#), [73](#), [84d](#), [85d](#), [105a](#), [106a](#), [186c](#), [201b](#), [216](#), [242g](#), [257b](#)
Eval: [88d](#), [281](#)

Eval.nstk: [88d](#)
Eval.p: [88d](#)
Eval.stk: [88d](#)
Eval.stksz: [88d](#)
Eval.str: [88d](#)
Eval (typedef): [281](#)
evalexpr(): [89a](#), [89b](#)
evalpostfix(): [89c](#)
expandprefix(): [51f](#), [248f](#)
fail(): [194c](#)
Fbuf: [323b](#), [323b](#)
Fbuf.buf: [323b](#)
Fbuf.lastfuzz: [323b](#)
Fbuf.lastln: [323b](#)
Fbuf.len: [323b](#)
Fbuf.lines: [323b](#)
Fbuf.mode: [323b](#)
Fbuf.nlines: [323b](#)
Fbuf (typedef): [323b](#)
Fchg: [323b](#), [323b](#)
Fchg.new: [323b](#)
Fchg.old: [323b](#)
Fchg.tmp: [323b](#)
Fchg (typedef): [323b](#)
fetch(): [295b](#), [296](#), [297b](#), [308b](#)
fetchbranch: [186c](#)
fetchpack(): [195c](#)
file: [61b](#)
findkey(): [199](#), [201a](#)
findref(): [201b](#)
findrepo(): [61e](#), [62a](#)
findroot: [62b](#), [118](#)
findroot(): [62b](#), [118](#)
findslashes(): [113e](#)
finish(): [312b](#), [319](#)
flushchanges(): [297b](#), [304a](#)
flushpkt(): [186c](#), [199](#), [209a](#), [214](#)
fmtcaps(): [186c](#), [195a](#)
fmtpkt(): [186c](#), [209b](#), [213b](#), [214](#), [216](#), [219b](#)
force: [199](#), [283a](#)
freediff(): [304b](#), [305a](#)
freemeta(): [171a](#), [173b](#)
freepatch(): [323a](#)
fullpath: [87b](#), [101a](#)
GBlob: [33a](#), [57c](#), [72b](#), [77b](#), [78a](#), [78b](#), [153b](#), [162a](#), [177d](#), [252c](#), [252d](#)
GCommit: [33a](#), [40a](#), [79a](#), [79e](#), [93b](#), [94c](#), [98e](#), [118](#), [144a](#), [144b](#), [146d](#), [152e](#), [161e](#), [165b](#), [177d](#), [194d](#), [219b](#), [252c](#),
[252d](#), [277](#)
gcommitgen(): [80c](#)
geartab-44: [155a](#), [266b](#)

genpack(): [171a](#), [172c](#)
GETBE16: [154c](#)
GETBE32: [30d](#), [37e](#), [38c](#), [44](#), [250](#), [273b](#)
GETBE64: [155c](#), [270](#)
gethead(): [212d](#), [214](#)
gitattach(): [66f](#)
Gिताux: [67b](#), [267](#)
Gिताux.crumb: [67b](#)
Gिताux.ncrumb: [67b](#)
Gिताux.ols: [67e](#)
Gिताux.olslast: [67e](#)
Gिताux.qdir: [85a](#)
Gिताux (typedef): [267](#)
gitclone(): [66f](#)
gitconnect(): [185d](#)
gitdestroyfid(): [66f](#)
gitdir: [70a](#), [75c](#)
gitdirmode: [48c](#), [61e](#), [81b](#), [81c](#)
githandshake(): [186a](#), [207b](#), [246b](#), [247b](#)
gitinit(): [61e](#), [100a](#)
gitmode(): [55b](#), [115c](#)
gitopen(): [66f](#)
gitread(): [66f](#)
gitsrv: [267](#)
gitstat(): [66f](#)
gitwalk1(): [66f](#)
GNone: [162c](#), [162d](#), [252c](#), [270](#)
G0delta: [33a](#), [172c](#), [177d](#), [252c](#)
grab(): [206c](#), [207a](#)
GRdelta: [33a](#), [172c](#), [177d](#), [252c](#)
groupname: [72a](#), [72b](#), [74d](#), [79b](#), [80c](#), [85c](#)
GTag: [47c](#), [72b](#), [77b](#), [80b](#), [81b](#), [161e](#), [165b](#), [177b](#), [177d](#), [252c](#), [252d](#)
GTree: [33a](#), [35c](#), [78c](#), [78d](#), [79b](#), [79e](#), [99](#), [100f](#), [145a](#), [153b](#), [173b](#), [177d](#), [236f](#), [252c](#), [252d](#), [282b](#)
gtreegen(): [79b](#)
Gxxx: [270](#)
HALFLONG-4: [292b](#), [292e](#)
Hash: [270](#), [270](#)
hash(): [154e](#), [168a](#), [168b](#)
Hash (typedef): [270](#)
hashcmp(): [161e](#), [180d](#)
hasheq(): [37e](#), [38c](#), [91b](#), [95a](#), [95b](#), [98e](#), [99](#), [145a](#), [178](#), [186c](#), [193](#), [199](#), [219a](#), [219b](#)
Hashsz: [30d](#), [155c](#), [161c](#)
hassuffix(): [284b](#)
hcompress(): [172c](#)
heads: [184b](#), [186c](#)
Hfmt(): [31b](#), [31c](#)
high-6: [292d](#), [292e](#)
hparse(): [31d](#), [50c](#), [51d](#), [77b](#), [84a](#), [180d](#), [186c](#), [192a](#), [199](#), [213b](#), [216](#), [258d](#), [259b](#), [274c](#)
Hunk: [323b](#), [323b](#)

Hunk.lnum: [323b](#)
Hunk.new: [323b](#)
Hunk.newcnt: [323b](#)
Hunk.newlen: [323b](#)
Hunk.newln: [323b](#)
Hunk.newpath: [323b](#)
Hunk.newsiz: [323b](#)
Hunk.old: [323b](#)
Hunk.oldcnt: [323b](#)
Hunk.oldlen: [323b](#)
Hunk.oldln: [323b](#)
Hunk.oldpath: [323b](#)
Hunk.oldsiz: [323b](#)
Hunk.orig: [323b](#)
Hunk.origlen: [323b](#)
Hunk.origsiz: [323b](#)
Hunk (typedef): [323b](#)
hunkcmp(): [315c](#), [316b](#)
hunkheader(): [313](#), [316b](#)
hwrite(): [52a](#), [163c](#), [172c](#), [173b](#), [176d](#)
idx: [40c](#)
idxcmp(): [41c](#)
Idxed: [102c](#), [285](#)
Idxed.n: [285](#)
Idxed (typedef): [285](#)
Idxent: [270](#), [285](#)
Idxent (typedef): [285](#)
idxsiz: [285](#)
indexed(): [105a](#), [106b](#)
indexpack(): [52a](#), [186c](#), [217b](#)
initconn(): [247b](#)
interactive: [39d](#), [163c](#), [165b](#), [170b](#), [172c](#), [176b](#), [186c](#), [250](#)
Internal-58: [66g](#), [85c](#)
intree: [104d](#), [105a](#)
isblank: [176a](#)
isdotordotdot(): [289b](#), [289c](#)
isindexed: [285](#), [285](#)
isloosedir(): [274a](#), [274c](#)
issmarthttp(): [242g](#)
isword(): [92a](#), [92b](#)
itemcmp(): [288](#), [289a](#)
Keep-18: [95e](#)
KiB: [270](#), [270](#)
lca(): [93d](#)
Lca-21: [93d](#), [94a](#), [95d](#)
Line: [287f](#), [287f](#)
Line.serial: [287f](#)
Line.value: [287f](#)
Line (typedef): [287f](#)

Lines-7: [297a](#), [297b](#), [298](#)
listonly: [186c](#)
listrefs(): [150e](#), [201b](#), [214](#)
loadent(): [104d](#), [105a](#)
loadpack(): [249d](#)
loadtree(): [152e](#), [153b](#), [173b](#)
loadwdir(): [104d](#), [105a](#)
localrepo(): [185d](#), [186b](#)
lockrepo(): [219b](#), [283b](#)
lookup(): [143e](#)
low-5: [292c](#), [292e](#)
lruhead: [33e](#), [34a](#), [152a](#), [249d](#)
lru tail: [33f](#), [34c](#), [152a](#), [249d](#)
Map: [283a](#)
Map.ours: [198c](#)
Map.ref: [198c](#)
Map.theirs: [198c](#)
Map (typedef): [283a](#)
matchcap(): [210b](#), [210c](#)
matchesfilter(): [142b](#), [144b](#), [146d](#)
matchesfilter1(): [143e](#), [144a](#), [144b](#)
matchpfx(): [51d](#), [51f](#)
max-2: [307a](#), [308b](#)
max-39: [163c](#), [256a](#)
Maxchunk-42: [155a](#), [266b](#)
MAXLINELEN: [287c](#), [293](#), [294b](#), [295b](#), [308a](#)
MAXPATHLEN: [287b](#), [289c](#), [290](#), [310b](#)
merge(): [308b](#)
Meta: [153c](#), [277](#)
Meta.delta: [152d](#)
Meta.head: [152d](#)
Meta.nchain: [152d](#)
Meta.ndelta: [152d](#)
Meta.off: [153d](#)
Meta.path: [277](#)
Meta.prev: [152d](#)
Meta (typedef): [277](#)
Metavec: [277](#)
Metavec.meta: [153c](#)
Metavec.metasz: [153c](#)
Metavec.nmeta: [153c](#)
Metavec (typedef): [277](#)
mflag: [296](#), [311b](#)
Mflg-13: [40b](#)
MiB: [270](#)
min-1: [306](#), [308b](#)
MIN-3: [291b](#), [293](#)
Minchunk-41: [155a](#), [266b](#)
mkcommit(): [118](#)

mkdir(): [217b](#), [221b](#)
mkols(): [76d](#), [77a](#)
mkoutpath(): [186c](#), [193](#)
mkpathname(): [289c](#), [290](#), [310b](#)
mktmpfile(): [310c](#), [311a](#)
mntpt: [66c](#), [66c](#)
mode: [289c](#), [296](#), [297b](#), [304a](#), [311b](#)
msgcount: [142b](#), [146d](#)
mstr: [108b](#), [108b](#)
murmurhash2(): [153b](#), [168b](#), [173b](#), [224](#)
namecmp(): [41d](#)
nbranch: [201b](#)
Nbranch-37: [279](#)
ncache: [152a](#), [249d](#)
NCACHE-11: [40b](#)
nchanged: [318c](#), [319](#), [322c](#), [323b](#)
newcand(): [300b](#), [301](#)
nextblk(): [154e](#), [155a](#), [167b](#)
nextline(): [144a](#)
nextqid: [69c](#)
nfile: [61c](#)
nheads: [186c](#)
Nhost-35: [185d](#), [206c](#), [279](#)
nidx: [42b](#)
Npackcache: [158d](#)
npacked: [202a](#)
npackf: [152c](#), [160a](#), [238f](#), [248f](#), [249d](#)
nparents: [115b](#), [118](#)
npath: [99](#), [100b](#), [100d](#), [100f](#)
Npath-36: [185d](#), [206c](#), [279](#)
Nport-34: [185d](#), [206c](#), [242d](#), [242e](#), [246g](#), [279](#)
Nproto-33: [185d](#), [206c](#), [279](#)
nrel: [100a](#)
nremoved: [201b](#)
nslash: [113e](#), [285](#)
nwdir: [105a](#), [285](#)
obj2dir(): [72a](#), [72b](#), [76d](#)
Objbuf: [282b](#), [282b](#)
Objbuf.dat: [55a](#)
Objbuf.hdr: [55a](#)
Objbuf.ndat: [55a](#)
Objbuf.nhdr: [55a](#)
Objbuf (typedef): [282b](#)
objbytes(): [282b](#)
objcache: [34a](#), [43b](#), [47c](#), [152a](#), [248b](#)
objcmp(): [43b](#), [176b](#)
Object: [270](#), [270](#)
Object (typedef): [270](#)
objectcrc(): [32c](#), [158c](#)

objgen(): [76c](#), [76d](#)
Objlist: [270](#), [270](#)
Objlist (typedef): [270](#)
Objq: [270](#), [270](#)
objq: [141f](#), [146d](#)
Objq (typedef): [270](#)
objread(): [76c](#), [84b](#), [85b](#)
Objset: [270](#), [270](#)
Objset (typedef): [270](#)
objwalk1(): [77b](#)
Ofmt(): [252b](#), [252d](#)
okref(): [190](#), [191](#), [217a](#)
okrefname(): [186c](#), [196c](#)
olsfree(): [68b](#), [71a](#)
olsnext(): [76d](#), [275](#)
olsreadloose(): [274c](#), [275](#)
olsreadpacked(): [274b](#), [275](#)
openpack(): [159c](#)
openpacks: [152c](#), [158d](#), [159a](#)
osadd(): [34a](#), [37e](#), [38b](#), [43b](#), [141f](#), [173b](#), [186c](#)
osclear(): [37d](#), [186c](#)
osfind(): [38c](#), [43b](#), [47c](#), [152e](#)
oshas(): [38d](#), [141f](#), [173b](#), [186c](#)
osinit(): [37c](#), [146d](#), [186c](#), [248b](#)
out: [142b](#), [143a](#), [145d](#), [259a](#), [272](#)
output(): [304a](#), [304b](#)
overlaps(): [307e](#), [308b](#)
Pack: [270](#)
Pack (typedef): [270](#)
Packf: [171b](#), [277](#)
packf: [45d](#), [152c](#), [158d](#), [159b](#), [159d](#), [160a](#), [238f](#), [248f](#), [249d](#)
Packf.idx: [277](#)
Packf.opentm: [46c](#)
Packf.pack: [46c](#)
Packf.refs: [46c](#)
Packf (typedef): [277](#)
packhdr(): [171c](#), [172c](#)
packoff(): [172c](#), [173a](#)
PaintMode: [95d](#)
PaintQcolor: [95e](#)
parent(): [90b](#), [93a](#)
parents: [118](#)
parse(): [316b](#)
parseauthor(): [155c](#), [258d](#)
parsecaps(): [186c](#), [199](#), [210b](#)
parsecmd(): [215b](#)
parsecommit(): [259b](#)
parseobject(): [45d](#), [48c](#), [157b](#), [277](#)
parseqid(): [191](#)

parsetag(): [47c](#), [48c](#)
parsetree(): [50b](#)
parseuri(): [185d](#), [206c](#)
Patch: [323b](#), [323b](#)
Patch.hunk: [323b](#)
Patch.name: [323b](#)
Patch.nhunk: [323b](#)
Patch (typedef): [323b](#)
path: [99](#), [100b](#), [100c](#), [100f](#)
pathcmp(): [106b](#), [106c](#)
pathfilt: [144a](#), [144b](#)
Pathmax: [47c](#), [270](#)
Pfilt: [272](#)
Pfilt.elc: [143a](#)
Pfilt.nsub: [143a](#)
Pfilt.show: [143a](#)
Pfilt.sub: [143a](#)
Pfilt (typedef): [272](#)
Pfmt(): [87b](#), [100b](#)
pfxmatch(): [108a](#)
pickdeltas(): [165b](#), [171a](#)
Pktmax: [192b](#), [195b](#), [195c](#), [209b](#), [217b](#), [219a](#), [283a](#), [283b](#)
pop(): [92e](#), [93a](#), [93d](#), [94c](#)
popcrumb(): [74b](#)
prefixed(): [186c](#), [194a](#), [194b](#)
prepare(): [293](#), [303](#)
printflg: [105a](#)
prune(): [299c](#), [303](#)
push(): [91c](#), [93a](#), [93d](#), [94c](#)
PUTBE16: [41a](#)
PUTBE32: [32a](#), [172c](#), [173b](#), [176d](#)
PUTBE64: [157a](#), [173b](#)
Qauthor-55: [66g](#), [81b](#), [82f](#), [83b](#), [83c](#)
Qbranch-47: [66g](#), [84b](#), [84c](#), [85c](#), [86a](#)
qclear(): [39f](#), [186c](#)
Qcommit-51: [66g](#), [78e](#), [79d](#), [81b](#), [86a](#)
Qcommitter-56: [66g](#), [81b](#), [82f](#), [83d](#)
Qctl-49: [66g](#), [74c](#), [75a](#)
QDIR: [72c](#), [73](#)
Qelt: [270](#), [270](#)
Qelt (typedef): [270](#)
Qfmt(): [252b](#), [253a](#)
Qhash-54: [66g](#), [81b](#), [82e](#), [82f](#), [83a](#)
Qhead-46: [66g](#), [81b](#), [82f](#), [83e](#)
qinit(): [39d](#), [146d](#), [186c](#)
Qmax-57: [66g](#)
Qmsg-52: [66g](#), [81a](#), [81b](#), [82b](#), [82d](#)
Qobject-48: [66g](#), [75b](#), [76a](#), [77b](#), [81b](#)
Qparent-53: [66g](#), [80a](#), [81b](#), [82a](#)

qpath(): [69c](#), [74d](#), [77b](#), [79b](#), [79e](#), [81b](#), [81c](#), [82a](#), [82d](#), [83a](#), [83c](#), [85c](#), [86a](#)
qpop(): [146d](#), [186c](#)
qput(): [40a](#), [141f](#), [146d](#)
qroot: [74d](#)
Qroot-45: [66g](#), [70a](#), [72c](#), [73](#), [74d](#)
Qtree-50: [66g](#), [76c](#), [77b](#), [81b](#), [81c](#)
queryexpr: [143a](#)
Qxxx: [66g](#)
range(): [295a](#)
Range-22: [94c](#), [95d](#)
readhash(): [292e](#), [293](#)
readidxobject(): [47c](#), [157b](#), [178](#)
readline(): [292a](#)
readloose(): [45d](#), [161e](#)
readobject(): [56c](#), [76d](#), [77b](#), [79b](#), [79e](#), [81b](#), [84a](#), [91e](#), [93a](#), [98e](#), [99](#), [100f](#), [118](#), [141f](#), [142b](#), [144b](#), [146d](#), [152e](#),
[157b](#), [165b](#), [172c](#), [173b](#), [186c](#), [194d](#), [199](#), [213b](#), [219b](#), [236f](#), [272](#)
readodelta(): [177d](#), [178](#)
readours(): [198d](#), [199](#)
readpacked(): [45d](#), [47c](#), [180e](#), [180e](#), [277](#)
readphase(): [186c](#), [199](#), [279](#)
readpkt(): [186c](#), [195b](#), [199](#), [209c](#), [213b](#), [216](#), [244b](#)
readrdelta(): [178](#)
readref(): [50c](#)
readrefdir(): [150d](#), [150e](#), [281](#)
readvint(): [46d](#), [47a](#)
recvnegotiate(): [215b](#), [217a](#)
recvpack(): [219b](#)
ref(): [35c](#), [36c](#), [70c](#), [73](#), [76d](#), [94a](#), [154e](#), [157b](#), [277](#)
refreshpacks(): [43b](#), [248f](#)
REGULAR_FILE: [287e](#), [290](#), [311a](#)
rejected(): [322a](#), [322c](#)
rejfd: [322a](#), [322c](#), [323b](#), [323b](#)
rejfile: [323b](#)
relapath: [108b](#), [113e](#)
removed: [201b](#)
rename(): [216](#)
repopath: [106a](#)
reporel(): [106a](#)
resolveref(): [118](#), [146d](#), [150e](#), [201b](#), [215a](#), [219b](#)
resolverefs(): [88b](#), [142b](#)
resolveremote(): [186c](#), [190](#), [192a](#)
reverse: [101f](#)
rflag: [290](#), [311b](#)
Rflg-12: [40b](#)
rootgen(): [74d](#)
rstr: [108b](#), [108b](#)
same(): [308a](#), [308b](#)
samedata(): [110](#)
sbread(): [186c](#), [195b](#)

scandir(): [289a](#), [289c](#)
search(): [300c](#), [301](#), [322c](#)
searchindex(): [238d](#), [238f](#), [248f](#)
searchln(): [321a](#)
Seed-38: [53c](#), [284b](#)
sendall: [201b](#)
sendpack(): [201b](#)
servelocal(): [185d](#), [207b](#)
servnegotiate(): [213a](#), [214](#)
servpack(): [213b](#)
shortlog: [140c](#), [144a](#)
show(): [144a](#)
showall: [63a](#), [63b](#)
showcommits(): [146d](#)
showconf(): [63a](#)
showdir(): [100e](#), [100f](#), [100f](#)
showprogress(): [39d](#), [165b](#), [172c](#), [250](#)
showquery(): [142b](#)
showrefs(): [213b](#), [215a](#), [216](#)
Sinit-28: [274b](#), [274c](#), [276a](#)
Siter-29: [274b](#), [274c](#), [276a](#)
Skip-20: [95e](#)
slurp(): [320](#), [322c](#)
slurpdir(): [77a](#), [85c](#), [150e](#), [151](#), [160b](#), [260](#)
sort(): [299a](#), [303](#)
Splitmask-43: [155a](#), [266b](#)
squishspace(): [294a](#), [294b](#)
staleidx: [109f](#), [110](#)
statfile(): [290](#), [311a](#)
stdout: [289c](#), [290](#), [295a](#), [295b](#), [296](#), [297b](#), [308b](#), [311b](#)
stone(): [301](#), [303](#)
stretch(): [167b](#), [168c](#)
strip: [176c](#), [323b](#)
strip(): [176c](#), [323b](#)
swapint(): [315d](#), [316b](#)
swapstr(): [315e](#), [316b](#)
take(): [89c](#), [90a](#)
Tfmt(): [252b](#), [252c](#)
Tinfo: [270](#), [270](#)
Tinfo (typedef): [270](#)
tmp-9: [310c](#), [311b](#)
TMPPATH-27: [150a](#)
tracepkt(): [208e](#), [209c](#), [254f](#)
tracked(): [57c](#)
trimhunk(): [316a](#), [316b](#)
Twixt-23: [164b](#)
Uflg-15: [105a](#)
unravel(): [302a](#), [303](#)
unref(): [34a](#), [71a](#), [74b](#), [76d](#), [81b](#), [98e](#), [99](#), [100f](#), [141f](#), [142b](#), [144b](#), [152e](#), [153b](#), [154d](#), [170b](#), [172c](#), [173b](#), [186c](#),

199, [213b](#), [219b](#), [249d](#), [269](#)
unsort(): [299b](#), [303](#)
updatepack(): [215b](#), [283b](#)
updaterefs(): [215b](#)
upstream: [184b](#), [184b](#), [192a](#)
Uqid: [267](#)
Uqid.idx: [68c](#)
Uqid.oid: [68c](#)
Uqid.ppath: [68c](#)
Uqid.t: [68c](#)
Uqid.uqid: [68c](#)
Uqid (typedef): [267](#)
uqidcache: [69b](#), [69c](#)
usage(): [229a](#)
Useragent-30: [243b](#), [244a](#)
username: [72a](#), [72b](#), [74d](#), [79b](#), [80c](#), [85c](#)
ustr: [285](#), [285](#)
validref(): [213a](#), [216](#)
WalkFlags: [40b](#)
walklink(): [236d](#)
wdir: [105a](#)
wdirpath: [106a](#)
wdirsz: [104a](#), [105a](#)
webclone(): [206c](#), [242g](#)
webopen(): [242g](#), [243b](#), [244b](#), [245c](#), [246f](#)
whichtmp-10: [310c](#), [311b](#)
word(): [90b](#), [92a](#)
workdir: [323b](#)
writeobj(): [56c](#), [57c](#), [118](#)
writeordercmp(): [172c](#), [173b](#)
writepack(): [171a](#), [199](#), [213a](#)
writephase(): [185b](#), [186c](#), [199](#)
writepkt(): [199](#), [208d](#), [208e](#), [209b](#)
writetree(): [55b](#)
zcommit-25: [91b](#), [95a](#), [95b](#), [281](#)
Zhash: [30b](#), [35c](#), [91b](#), [95a](#), [95b](#), [98e](#), [143e](#), [186c](#), [199](#), [201b](#), [219b](#)
_dprint(): [39b](#)
__anon_enum_11: [266b](#)
__anon_enum_1: [298](#)
__anon_enum_5: [55a](#)
__anon_enum_6: [276a](#)
__anon_enum_7: [279](#)

Bibliography

- [CS14] Scott Chacon and Ben Straub. *Pro Git*. 2014. Available at <https://git-scm.com/book/en/v2>. cited page(s) 14, 15
- [EJ01] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1), September 2001. Available at <http://www.ietf.org/rfc/rfc3174.txt>. cited page(s) 19
- [Gru86] Dick Grune. Concurrent versions system, a method for independent cooperation. Technical report, Vrije Universiteit, Amsterdam, 1986. Unpublished but available at https://dickgrune.com/Books/Publications/Concurrent_Versions_System,_a_method_for_independent_cooperation.ps. cited page(s) 13
- [Knu92] Donald E. Knuth. *Literate Programming*. Center for the Study of Language and Information, 1992. cited page(s) 15
- [KR88] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, 1988. cited page(s) 15
- [O’S09] Brian O’Sullivan. *Mercurial: The Definitive Guide*. O’Reilly, 2009. Available at <http://hgbook.red-bean.com/>. cited page(s) 14
- [Pad09] Yoann Padioleau. Syncweb, literate programming meets unison, 2009. <https://github.com/aryx/syncweb>. cited page(s) 15
- [Pad14] Yoann Padioleau. *Principia Softwarica: The Plan 9 Kernel 9*. 2014. cited page(s) 19, 24
- [Pad18] Yoann Padioleau. *Principia Softwarica: The Plan 9 Shell rc*. 2018. cited page(s) 13, 15
- [Pad25] Yoann Padioleau. *Principia Softwarica: The Plan 9 Utilities*. 2025. cited page(s) 15
- [Roc75] Marc J. Rochkind. The source code control system. *IEEE Transactions on Software Engineering*, pages 364–370, December 1975. Available at <http://www.basepath.com/aup/talks/SCCS-Slideshow.pdf>. cited page(s) 13
- [sha93] Secure hash standard, April 1993. Latest version available at <https://csrc.nist.gov/publications/detail/fips/180/4/final>. cited page(s) 19
- [Tic85] Walter F. Tichy. Rcs - a system for version control. *Software Practice and Experience*, pages 637–654, July 1985. Available at <http://www.gnu.org/s/rcs/tichy-paper.pdf>. cited page(s) 13