Git as I understand (1): Content-Addressable Object Model

I have used numerous source control systems in my past career. To list a few:

. ECMS: a proprietary version control tool used in AT&T/Lucent
. RCS: my personal hobby in my early Unix days
. CVS: the first source control I encountered in open source world
. Clearcase: When worked for enterprise customers, I used it a lot within Rational framwork
. SVN: Dominant tool after year 2005
. Mercurial: when I did low level hacking in JDK/jvm
. Git: When I began linux kernel trip

As a team player, I am very sensitive to any tool/work flow/process which can boost team productivity. That might explain my lasting interests in SCM system. Git is the last source control system I used as major tool. My first impression with Git are:

. It is tool developed by developer for developer
. With a succinct object model, the command line interface looks like not so well organized. I personally think "porcelain" interface is just a migration helper for users of existing SCMs. For me, its net effect is just prolonging my full adoption of Git's essence;
. The distributed model in Git provide just so much leeway for cooperation workflow, sometimes it confuses even veteran SCM users like me

All in all, Git is an unique and excellent source/content control system, founded by Linus' maverick design philosophy (Distributed Object Store/snapshot based). I personally think Git's biggest strength is its domain object model, and snapshot based history management. Once these two principles are established, any development of features never deviate from principles. Simple is beautiful, and consistency helps adoption.

There are more than plenty of good reference material for Git. There are many approaches to become a confident and competent Git user. My approach is bottom up, stick to the fundamental data object model, and using as few and as low level command lines (plumbing) as possible to achieve a workable SCM systems.

1. Principle and Rational
As Linus put it:

"In many ways you can just see git as a filesystem - it's content-addressable, and it has a notion of versioning, but I really really


Mac book pro 11.3 Linux…
The best way to configur…


Linux From Scratch for …
Ec2 linux server s… 1


Linux From Scratch for …
Linux Virtual Console(6)…

Linux From Scratch for …
dnscrypt-proxy in Archli…


Linux From Scratch for …
Git as I understand (9): …


Linux From Scratch for …
Linux Virtual Console(5)…


Linux From Scratch for …
Git as I understand (8): …


Linux From Scratch for …
Git as I understand (6): …


Linux From Scratch for …
Git as I understand (7): …


Linux From Scratch for …
Git as I understand (5): …


Linux From Scratch for …
Git as I understand (4): …


Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…


Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…


Regular Expression: A …
Mac book pro 11.3 Linux…


Automatic Scripts… 2
Mac book pro 11.3 Linux…


Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

As in Git manual:

```
GIT(1)

                Git Manual

                                        GIT(1)


NAME
       git - the stupid content tracker
```

So Git in Linus's eyes is a content-addressable file system with versioning support. Why is it "stupid"? My paraphrase for this model is as through a home-brewed "version control system":

a. There is demand for a new project, be it personal hobby or customer requirement;

b. I begins coding, and all my artifacts needs to be stored in a file system;

c. I use files to store my codes, and directory to organize my code;

d. I am a very logical man, and know the "split and conquer" strategy to handle complex problems. So I code step by step, with many meaningful interim results; When I am satisfied with an interim result, I archive all the content under root directory, and store it as a snapshot of my work; I generally name these snapshots with meaning directory names, for example, v0.1-add-feature-a, v0.2-add-feature-b,v0.3-fix-bug-c, etc, so I can remember what is my achievement in every snapshot; After sometimes, I have a lineage of snapshot history:

```
v0.1-->v0.2-->v0.3......
```

v0.1 is a directory tree, v0.2 is a directory tree, and so on. This ordered lineage reflects my incremental progress for my delivery.

e. When boss ask me what I have done today, I will show him with a diff commands:

```
diff -r v0.2 v0.3
```

To show him what file I have changed, what new files I have added, and what new directories I have added. the "diff" utility also help me remember what I have done between each interim results.

f. I am an efficient coder and have brain capability for parallel working. Sometimes I works on more that one idea in a project, and these ideas are quite separate from each other. I don't what to mess my codes with both ongoing ideas, so I have more than one copies of snapshots. I call them "working directory", and work each ideas on each "working directory":

```
.......--> v3.3---> idea-a (working directory)
                |
                |------> idea-b (working directory)
```

I switch between these working directories per my will. Someday, I think both ideas have been implemented and tested, I use a 3 party merge tool like "kdiff3" to merge 3 directory into one, and name it "v3.4":


GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…


GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1


GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…


GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…


GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …


Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…


GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …


QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …


Linux From Scratch for …
Git as I understand (7): …


Linux From Scratch for …
Git as I understand (5): …


Linux From Scratch for …
Git as I understand (4): …


Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…


Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…


Regular Expression: A …
Mac book pro 11.3 Linux…


Automatic Scripts… 2
Mac book pro 11.3 Linux…


Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
.......--> v3.3---> idea-a (working directory)------>v3.4
                |                                     ^


                |                                     |
                |------> idea-b (working directory)---|
```

g. Another developer, Tom, join this project. I tar all my snapshot directories and compressed it with gzip, bundled with one of my updated "concept diagram" and send the bundle file to Tom;

h. Tom is responsible for a specific feature "Tom-a", and I told him to base his work on "v3.4" directory. Tom finish his work on a directory "tom-a", and send back the result for me to review:

```
diff  v3.4 tom-a > tom-a.patch
```

After Tom create this "patch", he send it to me and an update "concept diagram graph":

```
.......--> v3.3---> idea-a (working directory)------>v3.4--->tom-a
                |                                     ^


                |                                     |
                |------> idea-b (working directory)---|
```

Then I review the patch file, and test Tom's work:

```
vi tom-a.patch
cp -r v3.4  review
cp tom-a.patch review
cd review
patch -p1 tom-a.patch
make
make test
...
```

We iterate step a through g on and on, until one day we all retired.

There are 3 critical concepts in this home-brewed "version control workflow",namely:

. file("blob" under git)

. working directory/tree: this directory we are working on ("working directory" under git)

. snapshot directory/tree: an archived working directory ("tree" under git)

. concept diagram: a working history logging what we have done ("commit log" under git), with multiple "lineage of working" ("branch" under git);

Such an home-brew workflow might seem "stupid", but from my point of view, Git just automates this workflow, and automate it very well. This might be one of the reason Linus call git a "stupid content tracker".

2. blob

blob store file contents, be it text or binary, without a file name. git will SHA-1 the blob content and use the SHA-1 hash value to store blob under ".git/object".

GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…

GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1

GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…

GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…

GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …

Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…

GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …

QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …

Linux From Scratch for …
Git as I understand (7): …

Linux From Scratch for …
Git as I understand (5): …

Linux From Scratch for …
Git as I understand (4): …

Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…

Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Regular Expression: A …
Mac book pro 11.3 Linux…

Automatic Scripts… 2
Mac book pro 11.3 Linux…

Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
.git
├── branches
├── config
├── description
├── HEAD
├── hooks
├── info
│   └── exclude
├── objects
│   ├── info
│   └── pack
└── refs
    ├── heads
    └── tags
[luke@rmbp project1]$ echo "Hello World!" > hello.txt
[luke@rmbp project1]$ git hash-object -t blob -w hello.txt
980a0d5f19a64b4b30a87d4206aade58726b60e3
[luke@rmbp project1]$ tree .git/objects/
.git/objects/
├── 98
│   └── 0a0d5f19a64b4b30a87d4206aade58726b60e3
├── info
└── pack

3 directories, 1 file
```

Here we add "hello.txt" into git object store as a blob. All objects are "zlib" deflated. We need a zlib command line tools:

http://www.zlib.net/zpipe.c [http://www.zlib.net/zpipe.c]
Since git use the compress level of "Z_BEST_SPEED", we need to make minor change to zpipe.c:

```
184     /* do compression if no arguments */
185     if (argc == 1) {
186 //        ret = def(stdin, stdout, Z_DEFAULT_COMPRESSION);
187 //        ret = def(stdin, stdout, Z_BEST_COMPRESSION);
188         ret = def(stdin, stdout, Z_BEST_SPEED);

gcc -lz zpipe.c -o zpipe


zpipe -d <.git/objects/98/0a0d5f19a64b4b30a87d4206aade58726b60e3 > blob
hexedit blob

00000000   62 6C 6F 62  20 31 33 00  48 65 6C 6C  6F 20 57 6F  blob 13.Hello Wo
00000010   72 6C 64 21  0A                                     rld!.
```

```
980a0d5f19a64b4b30a87d4206aade58726b60e3  blob
[luke@rmbp project1]$ zpipe <blob > blob.compress
[luke@rmbp              project1]$              cmp              blob.compress
.git/objects/98/0a0d5f19a64b4b30a87d4206aade58726b60e3
[luke@rmbp project1]$
```

So everything matches.
We can also use "git cat-file" to display content of a blob file:

```
[luke@rmbp project1]$ git cat-file -p 980a0d5f19a64b4b30a87d4206aade58726b60e3
Hello World!
```

3. tree and index/cache/staging area
We know to how to create blobs, it is time to organize our blobs(files) via tree(directory); First, let us examine an existing tree:

```
[luke@rmbp project1]$ git ls-tree -tr HEAD
040000 tree b6532456130e3225202f17e070c9b8178ef8d922 docs
100644 blob d9b401251bb36c51ca5c56c2ffc8a24a78ff20ae docs/README
100644 blob 9c36258b3007167a308da65fb6648e44f745da35 hello.c
100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 hello.txt
[luke@rmbp project1]$ tree
.
├── docs
│   └── README
├── hello.c
└── hello.txt

1 directory, 3 files
```

This tree corresponds to the working directory tree. Every entry in a tree has below format/content:

```
 <mode> SP <type> SP <object> TAB <file/path name>
```

Both tree and blob are stored as git objects. We can find them under ".git/objects":

```
[luke@rmbp project1]$ find .git/objects -type f  | sed -e 's$\.git/objects/$$g'
| sed 's$/$$g' | xargs -n1 git cat-file -p
tree 999ba42cf9a8a98eeb9e87d0f43955de7fd5fdf9
author Luke Luo <luke.jf.luo@gmail.com> 1400504866 +0800
committer Luke Luo <luke.jf.luo@gmail.com> 1400504866 +0800

Iniital Commit'
040000 tree b6532456130e3225202f17e070c9b8178ef8d922 docs
100644 blob 9c36258b3007167a308da65fb6648e44f745da35 hello.c
100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 hello.txt
100644 blob d9b401251bb36c51ca5c56c2ffc8a24a78ff20ae README
hello.c
```

GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…

GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1

GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…

GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…

GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …

Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…

GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …

QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …

Linux From Scratch for …
Git as I understand (7): …

Linux From Scratch for …
Git as I understand (5): …

Linux From Scratch for …
Git as I understand (4): …

Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…

Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Regular Expression: A …
Mac book pro 11.3 Linux…

Automatic Scripts… 2
Mac book pro 11.3 Linux…

Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
[luke@rmbp project1]$ git ls-tree 999ba42cf9a8a98eeb9e87d0f43955de7fd5fdf9
040000 tree b6532456130e3225202f17e070c9b8178ef8d922 docs
100644 blob 9c36258b3007167a308da65fb6648e44f745da35 hello.c

100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 hello.txt

[luke@rmbp 99]$ zpipe -d <9ba42cf9a8a98eeb9e87d0f43955de7fd5fdf9 > ~/tree.data

[luke@rmbp 99]$ hexedit ~/tree.data
00000000   74 72 65 65  20 31 30 33  00 34 30 30  30 30 20 64  tree 103.40000 d
00000010   6F 63 73 00  B6 53 24 56  13 0E 32 25  20 2F 17 E0  ocs..S$V..2% /..
00000020   70 C9 B8 17  8E F8 D9 22  31 30 30 36  34 34 20 68  p......"100644 h
00000030   65 6C 6C 6F  2E 63 00 9C  36 25 8B 30  07 16 7A 30  ello.c..6%.0..z0
00000040   8D A6 5F B6  64 8E 44 F7  45 DA 35 31  30 30 36 34  .._.d.D.E.510064
00000050   34 20 68 65  6C 6C 6F 2E  74 78 74 00  98 0A 0D 5F  4 hello.txt...._
00000060   19 A6 4B 4B  30 A8 7D 42  06 AA DE 58  72 6B 60 E3  ..KK0.}B...Xrk`.
```

All entries in a tree are either blobs or trees, and they must have been stored into object store before they can be listed an entry in a tree. This is because tree is only a mechanism to organize git objects, not normal files under working directory. tree is also an recursive data structure, starting from an root tree, you can include other low level "trees".

All tree must be created via the "index" mechanism provided by git. An "index" is an staging area to construct an tree object, and it is stored in "./git/index". "index","staging area", and "cache" are often used interchangeably in Git.

```
# empty the index/cache/staging area
[luke@rmbp project1]$ git read-tree --empty
# list index tree contents. Index tree is not stored into object yet, so you
can not use "git ls-tree"

[luke@rmbp project1]$ git ls-files --stage
# mode: git only track "644" or "755" for file and directory. "100" is normal
file, "120" is symbolic link, "040" is a directory
# we will re-use existing objects to construct an new tree object in index area
# add hello.c into index and use new file name "hello-new.c"
[luke@rmbp project1]$ git update-index --add --cacheinfo 100644
9c36258b3007167a308da65fb6648e44f745da35 hello-new.c
[luke@rmbp project1]$ git ls-files --stage
100644 9c36258b3007167a308da65fb6648e44f745da35 0 hello-new.c
# add hello.txt and use the path name "dir1/hello-new.txt"
[luke@rmbp project1]$ git update-index --add --cacheinfo 100644
980a0d5f19a64b4b30a87d4206aade58726b60e3 dir1/hello-new.txt
[luke@rmbp project1]$ git ls-files --stage
```


GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…


GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1


GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…


GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…


GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …


Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…


GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …


QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …


Linux From Scratch for …
Git as I understand (7): …


Linux From Scratch for …
Git as I understand (5): …


Linux From Scratch for …
Git as I understand (4): …


Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…


Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…


Regular Expression: A …
Mac book pro 11.3 Linux…


Automatic Scripts… 2
Mac book pro 11.3 Linux…


Github via SSH: a use c…
Mac book pro 11.3 Linux…


Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
100644 980a0d5f19a64b4b30a87d4206aade58726b60e3 0 dir1/hello-new.txt
100644 d9b401251bb36c51ca5c56c2ffc8a24a78ff20ae 0 docs-new/README

100644 9c36258b3007167a308da65fb6648e44f745da35 0 hello-new.c
# now we will store the index tree into object store and generate a tree object
[luke@rmbp project1]$ git write-tree
11a57232319f8e6bd9bd7031e284e1d6ed99c8db
[luke@rmbp project1]$ git ls-tree -tr 11a57232319f8e6bd9bd7031e284e1d6ed99c8db
040000 tree e54c875df80ddce5c11daa793779785f672146a2 dir1
100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 dir1/hello-new.txt
040000 tree b6532456130e3225202f17e070c9b8178ef8d922 docs-new
100644 blob d9b401251bb36c51ca5c56c2ffc8a24a78ff20ae docs-new/README
100644 blob 9c36258b3007167a308da65fb6648e44f745da35 hello-new.c
# here we can see a new tree "dir1" is created automatically for us since we
specify the path name "dir1/hello-new.txt"; "docs-new" has the same hash value
with original "docs" dir
# you can delete entries in index tree via "path/file name"

[luke@rmbp project1]$ git update-index --remove dir1/hello-new.txt
[luke@rmbp project1]$ git update-index --remove docs-new/README
```

Git does not track an empty directory. It is not possible to generate an tree object via an empty directory.
"git add" is a high level command which can add entry into index via the files in working directory. It is actually a two-steps command. First, it will generated blob and tree object using the files/dirs you added from working directory; then it will use "git update-index"/"git read-tree" to update the index. For example:

```
[luke@rmbp git]$ git init project1
Initialized empty Git repository in /tmp/git/project1/.git/
[luke@rmbp git]$ cd project1
[luke@rmbp project1]$ echo "Hello World!" > hello.txt
[luke@rmbp project1]$ tree .git/objects
.git/objects
├── info
└── pack

2 directories, 0 files
[luke@rmbp project1]$ git add hello.txt
[luke@rmbp project1]$ tree .git/objects
.git/objects
├── 98
│   └── 0a0d5f19a64b4b30a87d4206aade58726b60e3
├── info
└── pack

3 directories, 1 file
```

GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…

GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1

GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…

GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…

GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …

Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…

GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …

QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …

Linux From Scratch for …
Git as I understand (7): …

Linux From Scratch for …
Git as I understand (5): …

Linux From Scratch for …
Git as I understand (4): …

Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…

Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Regular Expression: A …
Mac book pro 11.3 Linux…

Automatic Scripts… 2
Mac book pro 11.3 Linux…

Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
├── 58
│   └── 1caa0fe56cf01dc028cc0b089d364993e046b6
├── 98
│   └── 0a0d5f19a64b4b30a87d4206aade58726b60e3
├── info
└── pack

4 directories, 2 files
[luke@rmbp project1]$ git ls-tree -tr 581caa0fe56cf01dc028cc0b089d364993e046b6
100644 blob 980a0d5f19a64b4b30a87d4206aade58726b60e3 hello.txt
```

Till now, Git has provides blob and tree to store our coding artifacts. The next logical step is to provide mechanism to record our "working history".

4. commit

Every snapshot of our work is a tree. We have tried to use meaningful names for archived snapshot trees in home-brewed version control system, and we also need to manually create a "concept diagram" to record our working history. In Git, "commit" is the key mechanism to record our working history. A commit must points to one and only one "tree object", since a commit corresponds to a snapshot. A commit also needs to have one or more "parent commit" since we need to record our "incremental/step by step" working history.

"git commit-tree" will generate a commit object. It mandate an existing tree object, and optional parent commit objects. It also mandate configuration data "author name" and "committer name".

```
[luke@rmbp project1]$ cat ~/.gitconfig
[user]
 email = luke.jf.luo@gmail.com

 name = Luke Luo
```

Normally, "author" and "committer" refer to same person. But there are exceptions, especially when merge and rebase happens. Here is an example to generate a commit to record a snapshot tree:

```
[luke@rmbp git]$ git init project1
Initialized empty Git repository in /tmp/git/project1/.git/
[luke@rmbp git]$ cd project1/
[luke@rmbp project1]$ echo "hello world" > hello.txt
[luke@rmbp project1]$ git hash-object -w hello.txt
3b18e512dba79e4c8300dd08aeb37f8e728b8dad
[luke@rmbp project1]$ git update-index --add hello.txt
[luke@rmbp project1]$ git ls-files -s
100644 3b18e512dba79e4c8300dd08aeb37f8e728b8dad 0 hello.txt
[luke@rmbp project1]$ git write-tree
68aba62e560c0ebc3396e8ae9335232cd93a3f60
```


GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…


GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1


GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…


GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…


GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …


Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…


GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …


QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …


Linux From Scratch for …
Git as I understand (7): …


Linux From Scratch for …
Git as I understand (5): …


Linux From Scratch for …
Git as I understand (4): …


Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…


Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…


Regular Expression: A …
Mac book pro 11.3 Linux…


Automatic Scripts… 2
Mac book pro 11.3 Linux…


Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
author Luke Luo <luke.jf.luo@gmail.com> 1400515839 +0800
committer Luke Luo <luke.jf.luo@gmail.com> 1400515839 +0800

Initial commit
[luke@rmbp                  project1]$                  zpipe                  -d
<.git/objects/68/aba62e560c0ebc3396e8ae9335232cd93a3f60 > commit.raw
[luke@rmbp project1]$ hexedit commit.raw
00000000   74 72 65 65  20 33 37 00  31 30 30 36  34 34 20 68  65 6C 6C 6F  2E
74 78 74  00 3B 18 E5  12 DB A7 9E  4C 83 00 DD  08 AE B3 7F  8E 72 8B 8D  tree
37.100644 hello.txt.;......L........r..

0000002C   AD
[luke@rmbp          project1]$          git          log          --pretty=fuller
0e44be6380e382d826d07046c2ad66f65752fc7b
commit 0e44be6380e382d826d07046c2ad66f65752fc7b
Author:     Luke Luo <luke.jf.luo@gmail.com>
AuthorDate: Tue May 20 00:10:39 2014 +0800
Commit:     Luke Luo <luke.jf.luo@gmail.com>
CommitDate: Tue May 20 00:10:39 2014 +0800

    Initial commit
[luke@rmbp project1]$
```

Here we can see "git commit-tree" did take "user" values from my "~/.gitconfig" to use them as "Author" and "committer" fields in commit; It also generate commit time stamp for us. Since this is the initial commit, I did not specify any "parent commit". The commit message (-m "Initial commit") is also recorded into commit object. "git log <commit>" is the canonical way to view the content of a commit and all its ancestor commits.

Next, we will add another file "README" into object store, create a new tree object containing both "hello.txt" and "README", and create a new commit based on this new tree, with our first commit as parent commit:

```
[luke@rmbp project1]$ echo "readme doc" > README
[luke@rmbp project1]$ git ls-files --stage
100644 3b18e512dba79e4c8300dd08aeb37f8e728b8dad 0 hello.txt
[luke@rmbp project1]$ git update-index --add README
[luke@rmbp project1]$ git ls-files --stage
100644 523f5db4d1652fb348f11b92417d7a2dc9dd7d70 0 README
100644 3b18e512dba79e4c8300dd08aeb37f8e728b8dad 0 hello.txt
[luke@rmbp project1]$ git write-tree
7927ca308b476e40989a7393983aacf23ee52026
[luke@rmbp    project1]$    git    commit-tree    -m    "second    commit"    -p
0e44be6380e382d826d07046c2ad66f65752fc7b
7927ca308b476e40989a7393983aacf23ee52026
b1f0d68803efd0d2fc620a8739ad9b8a7b822432
[luke@rmbp          project1]$          git          log          --pretty=fuller
b1f0d68803efd0d2fc620a8739ad9b8a7b822432
```


GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…


GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1


GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…


GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…


GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …


Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…


GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …


QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …


Linux From Scratch for …
Git as I understand (7): …


Linux From Scratch for …
Git as I understand (5): …


Linux From Scratch for …
Git as I understand (4): …


Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…


Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…


Regular Expression: A …
Mac book pro 11.3 Linux…


Automatic Scripts… 2
Mac book pro 11.3 Linux…


Github via SSH: a use c…
Mac book pro 11.3 Linux…


Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
    second commit

commit 0e44be6380e382d826d07046c2ad66f65752fc7b
Author:     Luke Luo <luke.jf.luo@gmail.com>
AuthorDate: Tue May 20 00:10:39 2014 +0800
Commit:     Luke Luo <luke.jf.luo@gmail.com>
CommitDate: Tue May 20 00:10:39 2014 +0800

    Initial commit
[luke@rmbp project1]$ git cat-file -p b1f0d68803efd0d2fc620a8739ad9b8a7b822432
tree 7927ca308b476e40989a7393983aacf23ee52026
parent 0e44be6380e382d826d07046c2ad66f65752fc7b
author Luke Luo <luke.jf.luo@gmail.com> 1400516794 +0800
committer Luke Luo <luke.jf.luo@gmail.com> 1400516794 +0800

second commit
[luke@rmbp project1]$ git ls-tree 7927ca308b476e40989a7393983aacf23ee52026
100644 blob 523f5db4d1652fb348f11b92417d7a2dc9dd7d70 README
100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad hello.txt

[luke@rmbp project1]$
```

Since one commit always points to exactly one tree. When we want to refer to a tree object, we can always do this indirectly via a corresponding commit object. tree and commits are all "tree-ish" under Git slang. For example, to take a look at the snapshot tree a commit stands for, we can do:

```
[luke@rmbp project1]$ git ls-tree -tr b1f0d68803efd0d2fc620a8739ad9b8a7b822432
(commit hash)
100644 blob 523f5db4d1652fb348f11b92417d7a2dc9dd7d70 README

100644 blob 3b18e512dba79e4c8300dd08aeb37f8e728b8dad hello.txt
# we can also import a commit's tree into index
[luke@rmbp project1]$ git read-tree --empty
[luke@rmbp project1]$ git read-tree b1f0d68803efd0d2fc620a8739ad9b8a7b822432
[luke@rmbp project1]$ git ls-files --stage
100644 523f5db4d1652fb348f11b92417d7a2dc9dd7d70 0 README

100644 3b18e512dba79e4c8300dd08aeb37f8e728b8dad 0 hello.txt
```

Git provide GUI tools to examine the working history ("concept diagram") embedded in commit objects. gitk is one of them:
[luke@rmbp project1]$ gitk b1f0d68803efd0d2fc620a8739ad9b8a7b822432

GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…

GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1

GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…

GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…

GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …

Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…

GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …

QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …

Linux From Scratch for …
Git as I understand (7): …

Linux From Scratch for …
Git as I understand (5): …

Linux From Scratch for …
Git as I understand (4): …

Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…

Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Regular Expression: A …
Mac book pro 11.3 Linux…

Automatic Scripts… 2
Mac book pro 11.3 Linux…

Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

[http://3.bp.blogspot.com/-oTTBFjrGOss/U3o0WiOIGgI/AAAAAAAABI0/R0iBmXY-s6U/s1600/gitk.png]

Thanks to git commit object, we don't need to draw "concept diagram" working history manually anymore.
In essence, we can regard commit as a special "annotation/metadata" information for a specific tree. A commit annotate a snapshot tree with specified "name/value" pairs like "parent commits","Author","committer","Author time","commit time", and "commit message". Via such annotations to snapshot tree, we can reconstruct a work history for all our archived snapshot tree.

"git commit" is a high level command comparing to "git commit-tree". It will first do a "git write-tree" according to the current content of index, then it will do "git commit-tree" to the newly generated tree.

5. tag
commit is a special annotation to a tree. Git also provides another type of object called "tag", which acts as generic annotation to commit objects. We use "git tag" to create a tag object:

```
[luke@rmbp project1]$ git log
commit d2f5de55ee3aa1dc90aec01a4e450f61285fece0
Author: Luke Luo <luke.jf.luo@gmail.com>
Date:   Tue May 20 15:27:36 2014 +0800

    initial commit
[luke@rmbp project1]$ tree .git
.git
├── branches
├── COMMIT_EDITMSG
├── config
├── description
├── HEAD
├── hooks
├── index
```


GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…


GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1


GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…


GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…


GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …


Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…


GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …


QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …


Linux From Scratch for …
Git as I understand (7): …


Linux From Scratch for …
Git as I understand (5): …


Linux From Scratch for …
Git as I understand (4): …


Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…


Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…


Regular Expression: A …
Mac book pro 11.3 Linux…


Automatic Scripts… 2
Mac book pro 11.3 Linux…


Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

```
|           └── master
├── objects
|   ├── 3b
|   |   └── 18e512dba79e4c8300dd08aeb37f8e728b8dad
|   ├── 68
|   |   └── aba62e560c0ebc3396e8ae9335232cd93a3f60
|   ├── d2
|   |   └── f5de55ee3aa1dc90aec01a4e450f61285fece0
|   ├── info
|   └── pack
└── refs
    ├── heads
    |   └── master
    └── tags

15 directories, 12 files
[luke@rmbp project1]$ git tag -a -m "tag for initial commit" v0.1
d2f5de55ee3aa1dc90aec01a4e450f61285fece0
[luke@rmbp project1]$ tree .git
.git
├── branches
├── COMMIT_EDITMSG
├── config
├── description
├── HEAD
├── hooks
├── index
├── info
|   └── exclude
├── logs
|   ├── HEAD
|   └── refs
|       └── heads
|           └── master
├── objects
|   ├── 3b
|   |   └── 18e512dba79e4c8300dd08aeb37f8e728b8dad
|   ├── 68
|   |   └── aba62e560c0ebc3396e8ae9335232cd93a3f60
|   ├── af
|   |   └── 6aae63b3551d4b650ea52408b4648d1c62ec71
|   ├── d2
|   |   └── f5de55ee3aa1dc90aec01a4e450f61285fece0
|   ├── info
|   └── pack
└── refs
    ├── heads
```

GRUB2 How To (6): PX…
Mac book pro 11.3 Linux…
The best way to configur…

GRUB2 How To (5): Buil…
Linux From Scratch for …
Ec2 linux server s… 1

GRUB2 How To (… 2
Linux From Scratch for …
Linux Virtual Console(6)…

GRUB2 How To (3) : UE…
Linux From Scratch for …
dnscrypt-proxy in Archli…

GRUB2 How To (… 4
Linux From Scratch for …
Git as I understand (9): …

Debugging with QEMU …
Linux From Scratch for …
Linux Virtual Console(5)…

GRUB2 How To (1) : Ma…
Linux From Scratch for …
Git as I understand (8): …

QEMU 1.5 build configur…
Linux From Scratch for …
Git as I understand (6): …

Linux From Scratch for …
Git as I understand (7): …

Linux From Scratch for …
Git as I understand (5): …

Linux From Scratch for …
Git as I understand (4): …

Linux From Scratch for …
Git as I understand (3): …

Linux From Scratch for …
Git as I understand (2): r…

Git as I understand (1): …

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Regular Expression: A …
Mac book pro 11.3 Linux…

Automatic Scripts… 2
Mac book pro 11.3 Linux…

Github via SSH: a use c…
Mac book pro 11.3 Linux…

Linux Virtual Console(3)…
Mac book pro 11.3 Linux…

Linux Virtual Console(2)…
Mac book pro 11.3 Linux…
How to debug system p…

af6aae63b3551d4b650ea52408b4648d1c62ec71

```
[luke@rmbp project1]$ git cat-file -p af6aae63b3551d4b650ea52408b4648d1c62ec71
object d2f5de55ee3aa1dc90aec01a4e450f61285fece0
type commit
tag v0.1
tagger Luke Luo <luke.jf.luo@gmail.com> 1400570948 +0800



tag for initial commit
[luke@rmbp                  project1]$                  zpipe                  -d
<.git/objects/af/6aae63b3551d4b650ea52408b4648d1c62ec71
tag 150object d2f5de55ee3aa1dc90aec01a4e450f61285fece0
type commit
tag v0.1
tagger Luke Luo <luke.jf.luo@gmail.com> 1400570948 +0800


tag for initial commit
```

We can see "git tag" create an object under ".git/objects", it also create a text file with file name of "tag name" under ".git/ref/tags/v0.1". The content of "v0.1" file is just the commit SHA1 value this tag SHA1 value. The data structure of a tag is quite simple. It contains the object hash it is pointing to, the type of pointed to object, the tag name, and tagger information. A tag also mandate a tag message.

Since there is one-one correspondence between tag and commit, and tag has a user friendly tag name, whenever we want to refer to a commit, we can refer to it indirectly via tag name:

```
[luke@rmbp project1]$ git log v0.1
commit d2f5de55ee3aa1dc90aec01a4e450f61285fece0
Author: Luke Luo <luke.jf.luo@gmail.com>
Date:   Tue May 20 15:27:36 2014 +0800

    initial commit
[luke@rmbp project1]$ git log d2f5de55ee3aa1dc90aec01a4e450f61285fece0
commit d2f5de55ee3aa1dc90aec01a4e450f61285fece0
Author: Luke Luo <luke.jf.luo@gmail.com>
Date:   Tue May 20 15:27:36 2014 +0800

    initial commit
```

6. Immutable Object

All Git objects are stored under ".git/ojbects" with path names formed from their SHA1 values. If you change any bit of the content of an object, the SHA1 values will change, and the new content will be stored with different path/file name. In theory, under Git, you can only create and delete object,

with different blob hash. So conceptually, we treat all these blobs as different versions of file "hello.txt". From the point of Git object store, it does not care the relationship among all unique objects. You throw an object to it, git store it, and return you a SHA1 as reference. You address you object with this reference. Since this reference is based on the content of you object, We can regard git as an "content-addressable object store".

Posted 20th May 2014 by Luke Luo

0 Add a comment

Enter your comment...

Comment as: bharathvenna.v Sign out

Publish Preview Notify me