SURVEY METHODOLOGY

SURVEY METHODOLOGY
This is the Subtitle

Robert M. Groves
Universitat de les Illes Balears

Floyd J. Fowler, Jr.
University of New Mexico

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Survey Methodology / Robert M. Groves . . . [et al.].
p. cm.—(Wiley series in survey methodology)
"Wiley-Interscience."
Includes bibliographical references and index.
ISBN 0-471-48348-6 (pbk.)
1. Surveys—Methodology. 2. Social sciences—Research—Statistical methods. I. Groves, Robert M. II. Series.

HA31.2.S873 2007
001.4'33—dc22 2004044064
Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my parents

CONTRIBUTORS

MASAYKI ABE, Fujitsu Laboratories Ltd., Fujitsu Limited, Atsugi, Japan

L. A. AKERS, Center for Solid State Electronics Research, Arizona State University, Tempe, Arizona

G. H. BERNSTEIN, Department of Electrical and Computer Engineering, University of Notre Dame, Notre Dame, South Bend, Indiana; formerly of Center for Solid State Electronics Research, Arizona State University, Tempe, Arizona

CONTENTS IN BRIEF

CONTENTS

LIST OF FIGURES

LIST OF TABLES

FOREWORD

This is the foreword to the book.

PREFACE

This is an example preface. This is an example preface. This is an example preface. This is an example preface.

R. K. WATTS

Durham, North Carolina
September, 2007

ACKNOWLEDGMENTS

From Dr. Jay Young, consultant from Silver Spring, Maryland, I received the initial push to even consider writing this book. Jay was a constant “peer reader” and very welcome advisor durying this year-long process.

To all these wonderful people I owe a deep sense of gratitude especially now that this project has been completed.

G. T. S.

ACRONYMS

ACGIH	American Conference of Governmental Industrial Hygienists
AEC	Atomic Energy Commission
OSHA	Occupational Health and Safety Commission
SAMA	Scientific Apparatus Makers Association

GLOSSARY

NormGibbs	Draw a sample from a posterior distribution of data with an unknown mean and variance using Gibbs sampling.
pNull	Test a one sided hypothesis from a numberically specified posterior CDF or from a sample from the posterior
sintegral	A numerical integration using Simpson's rule

SYMBOLS

A Amplitude
& Propositional logic symbol
a Filter Coefficient

$\mathcal{B}$ Number of Beats

INTRODUCTION

CATHERINE CLARK, PHD.

Harvard School of Public Health
Boston, MA, USA

The era of modern began in 1958 with the invention of the integrated circuit by J. S. Kilby of Texas Instruments [**?**]. His first chip is shown in Fig. I. For comparison, Fig. I.2 shows a modern microprocessor chip, [**?**].

This is the introduction. This is the introduction. This is the introduction. This is the introduction. This is the introduction. This is the introduction.

$$ABCDEF\alpha\beta\Gamma\Delta\sum_{def}^{abc} \tag{I.1}$$

REFERENCES

1. J. S. Kilby, "Invention of the Integrated Circuit," *IEEE Trans. Electron Devices,* **ED-23,** 648 (1976).
2. R. W. Hamming, *Numerical Methods for Scientists and Engineers*, Chapter N-1, McGraw-Hill, New York, 1962.
3. J. Lee, K. Mayaram, and C. Hu, "A Theoretical Study of Gate/Drain Offset in LDD MOSFETs" *IEEE Electron Device Lett.,* **EDL-7**(3). 152 (1986).

PART I

SUBMICRON SEMICONDUCTOR MANUFACTURE

CHAPTER 1

THE SUBMICROMETER SILICON MOSFET

The sheer volumne of answers can often stifle insight...The purpose of computing is insight, not numbers.

—Hamming [**?**]

1.1 Here is a normal section

Here is some text.

1.1.1 This is the subsection

Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text.

1.1.1.1 This is the subsubsection

Here is some text after the subsubsection. Here is some text after the subsubsection. Here is some text after the subsubsection. Here is some text after the subsubsection.

This is the paragraph Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text.

1.2 Tips On Special Section Heads

Here are some things you can do for a special section head.

1.3 Break Long Section heads with double backslash

Here is some normal text. Here is some normal text. Here is some normal text.

1.4 Here is a Section Title

See this section head for information on how to explicitly break lines in table of contents.

1.5 How to get lower case in section head: pH

Here is some normal text. Here is some normal text. Here is some normal text.

1.6 How to use a macro that has both upper and lower case parts: V_{Txyz}

See the top of this file where the definition and box were set.

1.7 Equation

For optimal vertical spacing, no blank lines before or after equations

$$\alpha\beta\Gamma\Delta \tag{1.1}$$

as you see here.

CHAPTER 2

FIRST EDITED BOOK SAMPLE CHAPTER TITLE

G. Alvarez and R. K. Watts

Carnegie Mellon University, Pittsburgh, Pennsylvania

2.1 Here is a normal section

Here is some text.

Survey Methodology, Second Edition.

CHAPTER 3

SECOND EDITED BOOK SAMPLE CHAPTER TITLE

GEORGE SMEAL, PH.D.[1], SALLY SMITH, M.D.[2] AND STANLEY KUBRICK[1]

[1]AT&T Bell Laboratories Murray Hill, New Jersey
[2]Harvard Medical School, Boston, Massachusetts

3.1 Sample Section

Here is some sample text.

Survey Methodology, Second Edition.

3.2 Example, Figure and Tables

EXAMPLE 3.1 Optional Example Name

Use Black's law [Equation (6.3)] to estimate the reduction in useful product life if a metal line is initially run at 55°C at a maximum line current density.

illustration here

Figure 3.1 Short figure caption.

Figure 3.2 Oscillograph for memory address access operations, showing 500 ps address access time and superimposed signals of address access in 1 kbit memory plane.

Table 3.1 Small Table

one	two	three	four
C	D	E	F

Table 3.2 Effects of the two types of $\alpha\beta \sum_B^A$ scaling proposed by Dennard and co-workers[a,b]

Parameter	κ Scaling	κ, λ Scaling
Dimension	κ^{-1}	λ^{-1}
Voltage	κ^{-1}	κ^{-1}
Currant	κ^{-1}	λ/κ^2
Dopant Concentration	κ	λ^2/κ

[a] Refs. 19 and 20.

[b] $\kappa, \lambda > 1$.

3.2.1 Side by Side Tables and Figures

Space for figure...

Figure 3.3 This caption will go on the left side of the page. It is the initial caption of two side-by-side captions.

Space for second figure...

Figure 3.4 This caption will go on the right side of the page. It is the second of two side-by-side captions.

The command `\sidebyside{}{}` works similarly for tables:

Table 3.3 Table Caption

one	two	three	four
a	little	sample	table

Table 3.4 Table Caption

A	B	C	D
a	second little	sample	table

When using `\sidebyside`, one must use the cross referencing command `\label{}` after and *outside* of `\caption{}`:

```
\begin{table}
\sidebyside{\caption{Table Caption}\label{tab1}
first table}
{\caption{Table Caption}\label{tab2} second table}
\end{table}
```

or,

```
\begin{figure}
\sidebyside{\vskip<dimen>\caption{fig caption}\label{fig1}}
{\vskip<dimen>\caption{fig caption}\label{fig2}}
\end{figure}
```

3.3 Algorithm

This is a sample algorithm.

Algorithm 3.1

state_transition algorithm {
for each neuron $j \in \{0, 1, \ldots, M-1\}$
{
calculate the weighted sum S_j using Eq. (6);
if ($S_j > t_j$)
{turn ON neuron; $Y_1 = +1$}
else if ($S_j < t_j$)
{turn OFF neuron; $Y_1 = -1$}
else
{no change in neuron state; y_j remains unchanged;}
}
}

Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text. Here is some normal text.

> This is a sample of extract or quotation. This is a sample of extract or quotation. This is a sample of extract or quotation.

1. This is the first item in the numbered list.
2. This is the second item in the numbered list. This is the second item in the numbered list. This is the second item in the numbered list.

- This is the first item in the itemized list.
- This is the first item in the itemized list. This is the first item in the itemized list. This is the first item in the itemized list.

  This is the first item in the itemized list.

  This is the first item in the itemized list. This is the first item in the itemized list. This is the first item in the itemized list.

PROBLEMS

3.1 For Hooker's data, Problem 1.2, use the Box and Cox and Atkinson procedures to determine a appropriate transformation of PRES in the regression of PRES on TEMP. find $\hat{\lambda}$, $\tilde{\lambda}$, the score test, and the added variable plot for the score. Summarize the results.

3.2 The following data were collected in a study of the effect of dissolved sulfur on the surface tension of liquid copper (Baes and Killogg, 1953).

x = Weight % sulfur	Y = Decrease in Surface Tension (dynes/cm), two Replicates	
0. 034	301	316
0. 093	430	422
0. 30	593	586

a) Find the transformations of X and Y sot that in the transformed scale the regression is linear.

b) Assuming that X is transformed to $\ln(X)$, which choice of Y gives better results, Y or $\ln(Y)$? (Sclove, 1972).

c) In the case of α_1?

d) In the case of α_2?

3.3 Examine the Longley data, Problem 3.3, for applicability of assumptions of the linear model.

3.4 In the case of Γ_1?

3.5 In the case of Γ_2?

EXERCISES

3.1 For Hooker's data, Exercise 1.2, use the Box and Cox and Atkinson procedures to determine a appropriate transformation of PRES in the regression of PRES on

TEMP. find $\hat{\lambda}$, $\tilde{\lambda}$, the score test, and the added variable plot for the score. Summarize the results.

3.2 The following data were collected in a study of the effect of dissolved sulfur on the surface tension of liquid copper (Baes and Killogg, 1953).

x = Weight % sulfur	Y = Decrease in Surface Tension (dynes/cm), two Replicates	
0. 034	301	316
0. 093	430	422
0. 30	593	586

a) Find the transformations of X and Y sot that in the transformed scale the regression is linear.

b) Assuming that X is transformed to $\ln(X)$, which choice of Y gives better results, Y or $\ln(Y)$? (Sclove, 1972).

c) In the case of Δ_1?

d) In the case of Δ_2?

3.3 Examine the Longley data, Problem 3.3, for applicability of assumptions of the linear model.

3.4 In the case of Γ_1?

3.5 In the case of Γ_2?

3.4 Summary

This is a summary of this chapter. Here are some references: [1], [4].

REFERENCES

1. J. S. Kilby, "Invention of the Integrated Circuit," *IEEE Trans. Electron Devices,* **ED-23,** 648 (1976).
2. R. W. Hamming, *Numerical Methods for Scientists and Engineers*, Chapter N-1, McGraw-Hill, New York, 1962.
3. J. Lee, K. Mayaram, and C. Hu, "A Theoretical Study of Gate/Drain Offset in LDD MOSFETs" *IEEE Electron Device Lett.,* **EDL-7**(3). 152 (1986).
4. A. Berenbaum, B. W. Colbry, D.R. Ditzel, R. D Freeman, and K.J. O'Connor, "A Pipelined 32b Microprocessor with 13 kb of Cache Memory," it Int. Solid State Circuit Conf., Dig. Tech. Pap., p. 34 (1987).

Appendix: This is the Chapter Appendix Title

This is an appendix with a title.

$$\alpha\beta\Gamma\Delta \tag{A.1}$$

Figure 3-A.1 This is an appendix figure caption.

Table 3-A.1 This is an appendix table caption

Date	Event
1867	Maxwell speculated the existence of electromagnetic waves.
1887	Hertz showed the existence of electromagnetic waves.
1890	Branly developed technique for detecting radio waves.
1896	Marconi demonstrated wireless telegraph.
1897	Marconi patented wireless telegraph.
1898	Marconi awarded patent for tuned communication.
1898	Wireless telegraphic connection between England and France established.

Appendix

This is a Chapter Appendix without a title.

Here is a math test to show the difference between using Computer Modern math fonts and MathTimes math fonts. When MathTimes math fonts are used the letters in an equation will match TimesRoman italic in the text. (*g, i, y, x, P, F, n, f, etc.*) Caligraphic fonts, used for $\mathcal{ABC}$ below, will stay the same in either case.

$$g_i(y|f) = \sum_x P(x|F_n) f_i(y|x) \mathcal{ABC} \tag{B.1}$$

where $g_i(y|F_n)$ is the function specifying the probability an object will display a value y on a dimension i given F_n the observed feature structure of all the objects.

CHAPTER 4

HOME

CHAPTER 5

BASIC CONCEPT

CHAPTER 6

ENVIRONTMENT SETUP

CHAPTER 7

LIFE CYCLE

CHAPTER 8

CREATE OPERATION

CHAPTER 9

CLONE OPERATION

CHAPTER 10

PERFORM CHANGES

MATERI GIT

Git Perform Changes

Dasar Git

Jadi, sebenarnya apa yang dimaksud dengan Git? Ini adalah bagian penting untuk dipahami, karena jika anda memahami apa itu Git dan cara kerja, maka dapat dipastikan anda dapat menggunakan Git secara efektif dengan mudah. Selama menerapkan Git, cobalah untuk menggantikan VCS lain yang mungkin sudah anda kenal sebelumnya, misalnya Subversion dan Perforce. Git sangat berbeda dengan sistem-sistem ini dalam hal simpan atau informasi yang digunakan, meski antar muka sangat mirip. Dengan memahami perbedaan ini diharapkan dapat membantu anda menghindari penggunaan saat menggunakan Git.

Snapshot, Bukan Perbedaan

Salah satu perbedaan yang mencolok antar Git dengan VCS lainnya (Subversion dan kawan-kawan) adalah dalam cara Git para datanya. Konsep konseptual,. Sis-

tem seperti ini (CVS, Subversion, Bazaar, dan yang lainnya) informasi yang tersimpannya sebagai sekumpulan berkas dan perubahan yang terjadi pada berkas-berkas tersebut,
Git dianggap datanya sebagai sebuah kumpulan snapshot dari sebuah miniatur sistem. Setiap kali anda melakukan komit, atau melakukan perubahan pada proyek Git anda, pada butir Git anda secara otomatis. Agar efisien, jika berkas tidak mengalami perubahan, Git tidak akan menyimpan file tersebut pada hanya pada file yang sama yang sebelumnya telah disimpan.

Git Punya Integritas
Segala sesuatu pada Git akan melalui proses checksum terlebih dahulu sebelum disimpan yang kemudian direferensikan oleh hasil checksum tersebut. Hal ini berarti tidak mungkin melakukan perubahan terhadap berkas manapun tanpa diketahui oleh Git. Fungsionalitas ini dimiliki oleh Git pada level terendahnya dan ini merupakan bagian tak terpisahkan dari filosofi Git. Anda tidak akan kehilangan informasi atau file yang tidak dimiliki oleh Git.

Mekanisme checksum yang digunakan oleh Git adalah SHA-1 hash. Ini merupakan sebuah susunan string yang terdiri dari 40 karakter heksadesimal (0 sampai 9 dan a sampai f) dan dihitung berdasarkan bentuk dari suatu berkas atau struktur pada pada Git. sebuah hash SHA-1 seperti berikut:
Anda akan melihat seperti ini pada berbagai tempat di Git. Faktanya, Git tidak memiliki nama file pada basisdatanya, pela nilai hash dari isi berkas.

Secara Umum Git Hanya Selesai Data
Bila anda melakukan operasi pada Git, hanya dari penambahan data pada basisdata Git. Sangat sulit membuat sistem melakukan sesuatu yang tidak bisa diurungkan atau membuatnya menghapus data dengan cara apa pun. Seperti pada berbagai VCS, anda bisa kehilangan atau mengacaukan perubahan yang belum di-commit; namun jika anda melakukan komit pada Git, akan sangat sulit kehilanngannya, apalagi jika anda secara teratur melakukan push basisdata anda pada repositori lain.
Hal ini membuat Git menyenangkan karena kita dapat berexperimen tanpa kehawatiran untuk mengacaukan proyek. Untuk lebih jelas dan dalam lagi tentang bagaimana Git menyimpan datanya dan bagaimana anda bisa mengembalikan yang hilang
Direktori Git adalah dimana Git menyimpan metadata dan database objek untuk projek anda. Ini adalah bahagian penting dari Git, dan inilah yang disalin saat anda melakukan kloning sebuah repositori dari komputer lain.
Direktori kerja adalah sebuah checkout tunggal dari satu versi dari projek. Fileberkas ini kemudian ditarik keluar dari basisdata yang terkompresi dalam direktori Git dan disimpan pada disk untuk anda pakai atau modifikasi.

Pementasan daerah adalah sebuah berkas sederhana, yang berada dalam direktori Git anda, yang mohon informasi mengenai apa yang menjadi komit selanjutnya. Ini disebut sebagai indeks, tapi semakin menjadi standar untuk maju sebagai area pementasan.

Alur kerja dasar Git adalah seperti ini:
Anda mengubah berkas dalam direktori kerja anda.
Anda membawa ke tahap, menambahkan snapshotnya ke area stage.
Anda melakukan komit, yang mengambil contoh seperti yang ada di daerah pementasan dan menyimpannya secara otomatis.
Jika sebuah versi tertentu dari sebuah berkas telah ada di direktori git, ia dianggap 'berkomitmen'. Jika berkas diubah (sudah diubah) maka sudah ditambahkan ke area stage, maka itu adalah 'staged'. Dan jika berkas telah diubah sejak terakhir dilakukan check out belum ditambahkan ke area stage maka itu adalah 'modified'. Pada Bab 2, anda akan lebih banyak membahas mengenai keadaan-keadaan ini dan bagaimana anda dapat memanfaatkan keadaan-keadaan yang bersangkutan dengan bagian 'bertahap'.

Seperti Ini Perform Changes

Jerry mengkloning repositori dan memutuskan untuk menerapkan operasi string dasar. Jadi dia menciptakan file string.c. Setelah menambahkan isinya, string.c akan terlihat seperti berikut:

```
#include ¡stdio.h¿

int my _strlen (char * s)
{
 char * p = s;

 sementara (* p)
   ++ p;

 return (p - s);
}

int main (void)
{
 int i;
 char * s [] =
  {
   "Git tutorial",
   "Tutorial Point"
  };

 untuk (i = 0; i ¡2; ++ i)
  printf ("panjang string % s = % d $n$ n", s [i], my _strlen (s [i]));

 kembali 0;
}
```

Dia menyusun dan menguji kodenya dan semuanya berjalan baik. Sekarang, dia bisa menambahkan perubahan ini ke repositori dengan aman.

Git menambahkan operasi menambahkan file ke area stage.

[jerry @ CentOS project] $ git status -s
?? tali
?? string.c

[jerry @ CentOS project] $ git add string.c
Git menunjukkan tanda tanya sebelum nama file. Jelas, file-file ini bukan bagian dari Git, dan karena itulah Git tidak tahu apa yang harus dilakukan dengan file-file ini. Itu sebabnya, Git menunjukkan tanda tanya sebelum nama file.

Jerry telah menambahkan file ke area penyimpanan, perintah status git akan menampilkan file yang ada di area stage.

[jerry @ CentOS project] $ git status -s
String.c
?? tali
Untuk melakukan perubahan, dia menggunakan perintah komit git diikuti dengan opsi -m. Jika kita menghilangkan opsi -m. Git akan membuka text editor dimana kita bisa menulis multiline commit message.

[jerry @ CentOS project] $ git commit -m 'Implementasikan fungsi my _strlen'
Perintah di atas akan menghasilkan hasil sebagai berikut:

[master cbe1249] Melaksanakan fungsi my _strlen
1 file berubah, 24 sisipan (+), 0 penghapusan (-)
buat mode 100644 string.c
Setelah berkomitmen untuk melihat rincian log, dia menjalankan perintah git log. Ini akan menampilkan informasi dari semua commit dengan commit komit mereka, commit author, commit date dan SHA-1 hash of commit.

[jerry @ CentOS project] $ git log
Perintah di atas akan menghasilkan hasil sebagai berikut:

melakukan cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Penulis: Jerry Mouse ¡jerry@tutorialspoint.com¿
Tanggal: Rabu Sep 11 08:05:26 2013 +0530

Diimplementasikan fungsi my _strlen

komit 19ae20683fc460db7d127cf201a1429523b0e319

Penulis: Tom Cat ¡tom@tutorialspoint.com¿
Tanggal: Rabu Sep 11 07:32:56 2013 +0530

Komit awal

Jerry memeriksa versi terbaru dari repositori dan mulai mengerjakan sebuah proyek.
Dia membuat file array.c di dalam direktori trunk.

[jerry @ CentOS ∼] $ cd project _repo / trunk /

[jerry @ CentOS trunk] $ cat array.c
Perintah di atas akan menghasilkan hasil berikut.

```
#include ¡stdio.h¿
#define MAX 16

int main (void) {
  int i, n, arr [MAX];
  printf ("Masukkan jumlah elemen:");
  scanf (" % d", & n);

  printf ("Enter the elements n n");

  untuk (i = 0; i ¡n; ++ i) scanf (" % d", & arr [i]);
  printf ("Array memiliki elemen berikut n n");
  untuk (i = 0; i ¡n; ++ i) printf (" | % d |", arr [i]);

  printf (" n n");
  kembali 0;
}
```

Dia ingin menguji kodenya sebelum melakukan.

[jerry @ CentOS trunk] $ buat array
cc array.c -o array

[jerry @ CentOS trunk] $ ./array
Masukkan jumlah elemen: 5
Masukkan elemen
1
2

3
4
5
Array memiliki elemen berikut
| 1 | | 2 | | 3 | | 4 | | 5 |
Dia menyusun dan menguji kodenya dan semuanya berjalan seperti yang diharapkan, sekarang saatnya melakukan perubahan.

[jerry @ CentOS trunk] status $ svn
? array.c
? array
Subversion menunjukkan '?' di depan nama file karena tidak tahu apa yang harus dilakukan dengan file-file ini.

Sebelum komit, Jerry perlu menambahkan file ini ke daftar perubahan yang tertunda.

[jerry @ CentOS trunk] $ svn tambahkan array.c
Sebuah array.c
Mari kita periksa dengan operasi 'status'. Subversion menunjukkan A sebelum array.c, artinya, file tersebut berhasil ditambahkan ke daftar perubahan yang tertunda.

[jerry @ CentOS trunk] status $ svn
? array
Sebuah array.c
Untuk menyimpan file array.c ke repositori, gunakan perintah komit dengan opsi -m diikuti oleh pesan komit. Jika Anda menghilangkan opsi -m, Subversion akan menampilkan editor teks tempat Anda bisa mengetikkan pesan multi-baris.

[jerry @ CentOS trunk] $ svn commit -m "Initial commit"
Menambahkan trunk / array.c
Mengirimkan data file
Komitmen revisi 2.
Sekarang file array.c berhasil ditambahkan ke repositori, dan nomor revisi bertambah satu.

CHAPTER 11

REVIEW CHANGES

MATERI GIT

Git Riview Changes

Dasar Git

Jadi, sebenarnya apa yang dimaksud dengan Git? Ini adalah bagian penting untuk dipahami, karena jika anda memahami apa itu Git dan cara kerja, maka dapat dipastikan anda dapat menggunakan Git secara efektif dengan mudah. Selama menerapkan Git, cobalah untuk menggantikan VCS lain yang mungkin sudah anda kenal sebelumnya, misalnya Subversion dan Perforce. Git sangat berbeda dengan sistem-sistem ini dalam hal simpan atau informasi yang digunakan, meski antar muka sangat mirip. Dengan memahami perbedaan ini diharapkan dapat membantu anda menghindari penggunaan saat menggunakan Git.

Snapshot, Bukan Perbedaan

Salah satu perbedaan yang mencolok antar Git dengan VCS lainnya (Subversion dan kawan-kawan) adalah dalam cara Git para datanya. Konsep konseptual,. Sis-

tem seperti ini (CVS, Subversion, Bazaar, dan yang lainnya) informasi yang tersimpannya sebagai sekumpulan berkas dan perubahan yang terjadi pada berkas-berkas tersebut,
Git dianggap datanya sebagai sebuah kumpulan snapshot dari sebuah miniatur sistem. Setiap kali anda melakukan komit, atau melakukan perubahan pada proyek Git anda, pada butir Git anda secara otomatis. Agar efisien, jika berkas tidak mengalami perubahan, Git tidak akan menyimpan file tersebut pada hanya pada file yang sama yang sebelumnya telah disimpan.

Git Punya Integritas
Segala sesuatu pada Git akan melalui proses checksum terlebih dahulu sebelum disimpan yang kemudian direferensikan oleh hasil checksum tersebut. Hal ini berarti tidak mungkin melakukan perubahan terhadap berkas manapun tanpa diketahui oleh Git. Fungsionalitas ini dimiliki oleh Git pada level terendahnya dan ini merupakan bagian tak terpisahkan dari filosofi Git. Anda tidak akan kehilangan informasi atau file yang tidak dimiliki oleh Git.

Mekanisme checksum yang digunakan oleh Git adalah SHA-1 hash. Ini merupakan sebuah susunan string yang terdiri dari 40 karakter heksadesimal (0 sampai 9 dan a sampai f) dan dihitung berdasarkan bentuk dari suatu berkas atau struktur pada pada Git. sebuah hash SHA-1 seperti berikut:
Anda akan melihat seperti ini pada berbagai tempat di Git. Faktanya, Git tidak memiliki nama file pada basisdatanya, pela nilai hash dari isi berkas.

Secara Umum Git Hanya Selesai Data
Bila anda melakukan operasi pada Git, hanya dari penambahan data pada basisdata Git. Sangat sulit membuat sistem melakukan sesuatu yang tidak bisa diurungkan atau membuatnya menghapus data dengan cara apa pun. Seperti pada berbagai VCS, anda bisa kehilangan atau mengacaukan perubahan yang belum di-commit; namun jika anda melakukan komit pada Git, akan sangat sulit kehilanngannya, apalagi jika anda secara teratur melakukan push basisdata anda pada repositori lain.
Hal ini membuat Git menyenangkan karena kita dapat berexperimen tanpa kehawatiran untuk mengacaukan proyek. Untuk lebih jelas dan dalam lagi tentang bagaimana Git menyimpan datanya dan bagaimana anda bisa mengembalikan yang hilang
Direktori Git adalah dimana Git menyimpan metadata dan database objek untuk projek anda. Ini adalah bahagian penting dari Git, dan inilah yang disalin saat anda melakukan kloning sebuah repositori dari komputer lain.
Direktori kerja adalah sebuah checkout tunggal dari satu versi dari projek. Fileberkas ini kemudian ditarik keluar dari basisdata yang terkompresi dalam direktori Git dan disimpan pada disk untuk anda pakai atau modifikasi.

Pementasan daerah adalah sebuah berkas sederhana, yang berada dalam direktori Git anda, yang mohon informasi mengenai apa yang menjadi komit selanjutnya. Ini disebut sebagai indeks, tapi semakin menjadi standar untuk maju sebagai area pementasan.

Alur kerja dasar Git adalah seperti ini:
Anda mengubah berkas dalam direktori kerja anda.
Anda membawa ke tahap, menambahkan snapshotnya ke area stage.
Anda melakukan komit, yang mengambil contoh seperti yang ada di daerah pementasan dan menyimpannya secara otomatis.
Jika sebuah versi tertentu dari sebuah berkas telah ada di direktori git, ia dianggap 'berkomitmen'. Jika berkas diubah (sudah diubah) maka sudah ditambahkan ke area stage, maka itu adalah 'staged'. Dan jika berkas telah diubah sejak terakhir dilakukan check out belum ditambahkan ke area stage maka itu adalah 'modified'. Pada Bab 2, anda akan lebih banyak membahas mengenai keadaan-keadaan ini dan bagaimana anda dapat memanfaatkan keadaan-keadaan yang bersangkutan dengan bagian 'bertahap'.

Seperti Ini Review Changes

Setelah melihat rincian komit, Jerry menyadari bahwa panjang string tidak boleh negatif, karena itulah dia memutuskan untuk mengubah jenis fungsi my _strlen yang kembali.

Jerry menggunakan perintah git log untuk melihat detail log.

$ git log
Perintah di atas akan menghasilkan hasil berikut.

melakukan cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Penulis: Jerry Mouse ¡jerry@tutorialspoint.com¿
Tanggal: Rabu Sep 11 08:05:26 2013 +0530
Diimplementasikan fungsi my _strlen

Jerry menggunakan perintah git show untuk melihat rincian komit. Perintah git show mengambil SHA-1 commit ID sebagai parameter.

$ git show cbe1249b140dad24b2c35b15cc7e26a6f02d2277

Perintah di atas akan menghasilkan hasil sebagai berikut:

melakukan cbe1249b140dad24b2c35b15cc7e26a6f02d2277
Penulis: Jerry Mouse ¡jerry@tutorialspoint.com¿
Tanggal: Rabu Sep 11 08:05:26 2013 +0530

Diimplementasikan fungsi my _strlen

diff - git a / string.c b / string.c

```
mode file baru 100644
indeks 0000000..187afb9
        — / dev / null

+++ b / string.c
@@ -0,0 +1,24 @@
+ # include ¡stdio.h¿
+
+ int my _strlen (char * s)
+ {
 +
 char * p = s;
 +
 +
 sementara (* p)
 + ++ p;
 + return (p -s);
 +
}
+
```

Dia mengubah jenis fungsi kembali dari int menjadi size _t. Setelah menguji kode tersebut, dia mengulas perubahannya dengan menjalankan perintah diff git.

```
$ git diff
```

Perintah di atas akan menghasilkan hasil sebagai berikut:

```
diff - git a / string.c b / string.c
indeks 187afb9..7da2992 100644
— a / string.c
+++ b / string.c
@@ -1,6 +1,6 @@
#include ¡stdio.h¿

-int my _strlen (char * s)
+ size _t my _strlen (char * s)
{
 char * p = s;
 @@ -18,7 +18,7 @@ int main (void)
};
untuk (i = 0; i ¡2; ++ i)
{
 - printf (”panjang string % s = % d n n”, s [i], my _strlen (s [i]));
 + printf (”panjang string % s = % lu n n”, s [i], my _strlen (s [i]));
```

```
 kembali 0;
}
```

Git diff menunjukkan tanda '+' sebelum baris, yang baru ditambahkan dan '-' untuk baris yang dihapus.

Melihat Sejarah Komit

Setelah Anda membuat beberapa commit, atau jika Anda telah mengkloning sebuah repositori dengan riwayat komit yang ada, Anda mungkin ingin melihat kembali untuk melihat apa yang telah terjadi. Alat yang paling dasar dan ampuh untuk melakukan ini adalah perintah git log.

Contoh-contoh ini menggunakan proyek sederhana yang disebut "simplegit". Untuk mendapatkan proyek, jalankan

git klon https://github.com/schacon/simplegit-progit

Saat Anda menjalankan git log dalam proyek ini, Anda harus mendapatkan keluaran yang terlihat seperti ini:

```
$ git log
melakukan ca82a6dff817ec66f44342007202690a93763949
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sen 17 Mar 21:52:11 2008 -0700

  mengubah nomor versinya

komit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sat 15 Mar 16:40:33 2008 -0700
lepaskan tes yang tidak perlu
lakukan a11bef06a3f659402fe7563abf99ad00de2209e6
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sat Mar 15 10:31:28 2008 -0700

 komit pertama
```

Secara default, tanpa argumen, git log mencantumkan komit yang dibuat di repositori tersebut dalam urutan kronologis terbalik - yaitu, komit terbaru muncul lebih dulu. Seperti yang dapat Anda lihat, perintah ini mencantumkan masing-masing komit dengan checksum SHA-1, nama penulis dan e-mail, tanggal penulisan, dan pesan komit.

Sejumlah besar dan berbagai pilihan untuk perintah git log tersedia untuk menunjukkan dengan tepat apa yang Anda cari. Di sini, kami akan menunjukkan beberapa yang paling populer.

Salah satu pilihan yang lebih bermanfaat adalah -p, yang menunjukkan perbedaan yang diperkenalkan pada masing-masing komit. Anda juga bisa menggunakan -2, yang membatasi output hanya pada dua entri terakhir:

```
$ git log -p -2
```

```
melakukan ca82a6dff817ec66f44342007202690a93763949
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sen 17 Mar 21:52:11 2008 -0700

  mengubah nomor versinya

diff - git a / Rakefile b / Rakefile
indeks a874b73..8f94139 100644
— a / Rakefile
+++ b / Rakefile
@@ -5,7 +5,7 @@ memerlukan 'rake / gempackagetask'
spec = Gem :: Specification.new do | s |
   s.platform = Gem :: Platform :: RUBY
   s.name = "simplegit"
- s.version = "0.1.0"
+ s.version = "0.1.1"
   s.author = "Scott Chacon"
   s.email = "schacon@gee-mail.com"
   s.summary = "Sebuah permata sederhana untuk menggunakan Git dalam
kode Ruby."

komit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sat 15 Mar 16:40:33 2008 -0700

  lepaskan tes yang tidak perlu

                diff - git a / lib / simplegit.rb b / lib / simplegit.rb
                indeks a0a60ae..47c6340 100644
                — a / lib / simplegit.rb
                +++ b / lib / simplegit.rb
                @@ -18,8 +18,3 @@ kelas SimpleGit
                  akhir
akhir
-
-if $ 0 == _ _FILE _ _
- git = SimpleGit.new
- menempatkan git.show
```

-akhir
n Tidak ada baris baru di akhir file

Pilihan ini menampilkan informasi yang sama namun dengan diff langsung mengikuti setiap entri. Ini sangat membantu untuk tinjauan kode atau untuk menelusuri dengan cepat apa yang terjadi selama serangkaian komit yang telah ditambahkan oleh kolaborator. Anda juga bisa menggunakan serangkaian opsi merangkum dengan log git. Misalnya, jika Anda ingin melihat beberapa statistik singkat untuk setiap komit, Anda dapat menggunakan opsi –stat:

$ git log –stat
melakukan ca82a6dff817ec66f44342007202690a93763949
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sen 17 Mar 21:52:11 2008 -0700

mengubah nomor versinya

Rakefile | 2 + -
1 file berubah, 1 insertion (+), 1 deletion (-)

komit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sat 15 Mar 16:40:33 2008 -0700

lepaskan tes yang tidak perlu

lib / simplegit.rb | 5 ——
1 file berubah, 5 deletions (-)
lakukan a11bef06a3f659402fe7563abf99ad00de2209e6

Penulis: Scott Chacon ¡schacon@gee-mail.com¿
Tanggal: Sat Mar 15 10:31:28 2008 -0700
komit pertama
README | 6 ++++++
Rakefile | 23 +++++++++++++++++++++
lib / simplegit.rb | 25 +++++++++++++++++++++++++
3 file berubah, 54 sisipan (+)
Sekian Materi Tentang Review Changes Ini Semoga Bermanfaat.

CHAPTER 12

COMMIT CAHNGES

CHAPTER 13

PUSH OPERATION

CHAPTER 14

UPDATE OPERATION

Git adalah version control system yang digunakan para developer untuk mengembangkan software secara bersama-bersama. Fungsi utama git yaitu mengatur versi dari source code program anda dengan mengasih tanda baris dan code mana yang ditambah atau diganti.

Git ini sebenernya memudahkan programmer untuk mengetahui perubahan source codenya daripada harus membuat file baru seperti Program.java, ProgramRevisi.java, ProgramRevisi2.java, ProgramFix.java. Selain itu, dengan git kita tak perlu khawatir code yang kita kerjakan bentrok, karena setiap developer bias membuat branch sebagai workspacenya.Fitur yang tak kalah keren lagi, pada git kita bisa memberi komentar pada source code yang telah ditambah/diubah, hal ini mempermudah developer lain untuk tahu kendala apa yang dialami developer lain. Untuk mengetahui

bagaimana menggunakan git, berikut perintah-perintah dasar git:

1. Git init : untuk membuat repository pada file lokal yang nantinya ada folder .git
2. Git status : untuk mengetahui status dari repository lokal
3. Git add : menambahkan file baru pada repository yang dipilih

4. Git commit : untuk menyimpan perubahan yang dilakukan, tetapi tidak ada perubahan pada remote repository.
5. Git push : untuk mengirimkan perubahan file setelah di commit ke remote repository.
6. Git branch : melihat seluruh branch yang ada pada repository
7. Git checkout : menukar branch yang aktif dengan branchyang dipilih
8. GIt merge : untuk menggabungkan branch yang aktif dan branch yang dipilih
9. Git clone : membuat Salinan repository lokal

Contoh dari software version control system adalah github, bitbucket, snowy evening, dan masih banyak lagi. Jika anda sebagai developer belum mengetahui fitur git ini, maka anda wajib mencoba dan memakainya. Karena banyak manfaat yang akan didapat dengan git ini.

Dalam melakukan pemrograman, perubahan spesifikasi atau kebutuhan adalah hal yang tidak dapat dihindari. Tidak ada program yang dapat dituliskan dengan sempurna pada percobaan pertama. Hal ini menyebabkan pengembang perangkat lunak sangat dekat dengan sistem kontrol versi, baik secara manual maupun menggunakan perangkat lunak khusus. Seri tulisan ini akan membahas tentang sistem kontrol versi, kegunaannya, serta contoh kasus menggunakan git, salah satu perangkat lunak populer untuk kontrol versi.

Dasar Kontrol Versi

Kegunaan utama dari sistem kontrol versi ialah sebagai alat untuk manajemen kode program. Terdapat dua kegunaan utama dari sistem ini, yaitu:

1. Menyimpan versi lama dari kode, maupun
2. Menggabungkan perubahan-perubahan kode dari versi lama (misal: untuk mengembalikan fitur yang telah dihapus) ataupun menggabungkan perubahan dari orang lain (misal: menggabungkan fitur yang dikembangkan oleh anggota tim lain).

Tanpa menggunakan sistem kontrol versi, yang sering saya temukan (dan dulunya saya gunakan, sebelum mengetahui tentang kontrol versi) ialah pengunaan direktori untuk memisahkan beberapa versi program. Sistem kontrol versi, seperti git, hg,

atau bzr, dikembangkan untuk menyelesaikan masalah-masalah di atas. Karena tidak ingin membahas terlalu banyak, artikel ini hanya akan menjelaskan pengunaan git, karena kelihatannya git merupakan perangkat lunak kontrol versi yang paling populer untuk sekarang (mengingat popularitas Github dan pengunaan git pada kernel Linux).

Git

Git adalah sebuah perangkat lunak untuk mengontrol versi sebuah perangkat lunak "VCS/Version Control System". Git diciptakan oleh Linux Torvalds, yang pada awalnya ditujukan untuk pengembangan kernel Linux. Saat ini banyak perangkat lunak yang terkenal menggunakan Git sebagai pengotrol revisinya. Pada bab ini akan

mempelajari bagaimana cara menggunakan Git seperti proses life cycle Git, operasi-operasi dasar dan bagaimana cara menangani masalah saat menggunakan Git. Git

menyimpan sementara perubahan yang telah di buat pada copy-an pekerjaan Anda sehingga Anda dapat mengerjakan sesuatu yang lain, lalu kembali dan terapkan kembali nanti. Stashing berguna jika Anda perlu mengubah konteks dan mengerjakan hal lain dengan lebih cepat, tapi Anda sedang melewati perubahan kode dan tidak cukup siap untuk melakukannya. Perintah git stash mengambil perubahan yang tidak terikat

(baik yang dipasang maupun yang tidak terpasang), menyimpannya untuk penggunaan selanjutnya, lalu mengembalikannya dari salinan pekerjaan Anda. Sebagai contoh: $ git status

On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
$ git stash
Saved working directory and index state WIP on master: 5002d47 our new homepage
HEAD is now at 5002d47 our new homepage
$ git status
On branch master
nothing to commit, working tree clean

Pada poin ini Anda bebas melakukan perubahan, membuat commit baru, mengganti cabang, dan melakukan operasi Git lainnya; Kemudian kembali dan pasang kembali simpanan Anda saat Anda siap.

1. Perhatikan bahwa simpanannya adalah lokal ke tempat penyimpanan Git Anda.
2. Mengajukan kembali perubahan tersimpan Anda
3. Anda dapat mengajukan permohonan kembali sebelumnya menyimpan perubahan dengan git stash pop:

$ git status
On branch master
nothing to commit, working tree clean
$ git stash pop

On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)

Memindahkan simpanan Anda akan menghilangkan perubahan dari simpanan Anda dan memasangnya kembali ke salinan pekerjaan Anda.

Sebagai alternatif, Anda dapat mengajukan permohonan kembali perubahan pada copy pekerjaan Anda dan menyimpannya di tempat penyimpanan dengan git stash berlaku:

$ git stash apply
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html

Ini berguna jika Anda ingin menerapkan perubahan tersimpan yang sama ke beberapa cabang.

Sekarang setelah Anda mengetahui dasar-dasar stashing, ada satu peringatan dengan penyimpanan git yang perlu Anda sadari: Secara default Git tidak akan menyimpan perubahan yang dibuat pada file yang tidak terlacak atau diabaikan.

Menyembunyikan file yang tidak terlacak atau diabaikan

Secara default, menjalankan git stash akan menyimpan:

1. perubahan yang telah ditambahkan ke indeks Anda (perubahan bertahap)
2. Perubahan yang dilakukan pada file yang saat ini dilacak oleh Git (perubahan yang tidak terhapus)

Tapi itu tidak akan disimpan:

1. file baru dalam copy pekerjaan Anda yang belum dipentaskan
2. file yang telah diabaikan

Jadi jika kita menambahkan file ketiga ke contoh kita di atas, tapi jangan tingkatkan (misal kita tidak menjalankan git add), git stash tidak akan menyimpannya.

$ script.js
$ git status
On branch master
Changes to be committed:
new file: style.css
Changes not staged for commit:
modified: index.html
Untracked files:

script.js
$ git stash
Saved working directory and index state WIP on master: 5002d47 our new homepage
HEAD is now at 5002d47 our new homepage
$ git status
On branch master
Untracked files:
script.js
Menambahkan opsi -u (atau –include-unracked) memberitahu git stash untuk juga menyimpan file yang tidak terlacak.

Jadi, sebenarnya apa yang dimaksud dengan Git? Ini adalah bagian penting untuk dipahami, karena jika anda memahami apa itu Git dan cara kerjanya, maka dapat dipastikan anda dapat menggunakan Git secara efektif dengan mudah. Selama mempelajari Git, cobalah untuk melupakan VCS lain yang mungkin telah anda kenal sebelumnya, misalnya Subversion dan Perforce. Git sangat berbeda dengan sistem-sistem tersebut dalam hal menyimpan dan memperlakukan informasi yang digunakan, walaupun antar-muka penggunanya hampir mirip. Dengan memahami perbedaan tersebut diharapkan dapat membantu anda menghindari kebingungan saat menggunakan Git.

Salah satu perbedaan yang mencolok antar Git dengan VCS lainnya (Subversion dan kawan-kawan) adalah dalam cara Git memperlakukan datanya. Secara konseptual, kebanyakan sistem lain menyimpan informasi sebagai sebuah daftar perubahan berkas. Sistem seperti ini (CVS, Subversion, Bazaar, dan yang lainnya) memperlakukan informasi yang disimpannya sebagai sekumpulan berkas dan perubahan yang terjadi pada berkas-berkas tersebut.

Git memperlakukan datanya sebagai sebuah kumpulan snapshot dari sebuah miniatur sistem berkas. Setiap kali anda melakukan commit, atau melakukan perubahan pada proyek Git anda, pada dasarnya Git merekam gambaran keadaan berkas-berkas anda pada saat itu dan menyimpan referensi untuk gambaran tersebut. Agar efisien, jika berkas tidak mengalami perubahan, Git tidak akan menyimpan berkas tersebut melainkan hanya pada file yang sama yang sebelumnya telah disimpan.

Ini adalah sebuah perbedaan penting antara Git dengan hampir semua VCS lain. Hal ini membuat Git mempertimbangkan kembali hampir setiap aspek dari version control yang oleh kebanyakan sistem lainnya disalin dari generasi sebelumnya. Ini membuat Git lebih seperti sebuah miniatur sistem berkas dengan beberapa tool yang luar biasa ampuh yang dibangun di atasnya, ketimbang sekadar sebuah VCS. Kita akan mempelajari beberapa manfaat yang anda dapatkan dengan memikirkan data anda dengan cara ini ketika kita membahas ”Git branching” pada Bab 3.

Hampir Semua Operasi Dilakukan Secara Lokal

Kebanyakan operasi pada Git hanya membutuhkan berkas-berkas dan resource lokal - tidak ada informasi yang dibutuhkan dari komputer lain pada jaringan anda. Jika Anda terbiasa dengan VCS terpusat dimana kebanyakan operasi memiliki overhead latensi jaringan, aspek Git satu ini akan membuat anda berpikir bahwa para dewa kecepatan telah memberkati Git dengan kekuatan. Karena anda memiliki seluruh sejarah dari proyek di lokal disk anda, dengan kebanyakan operasi yang tampak hampir seketika.

Sebagai contoh, untuk melihat history dari proyek, Git tidak membutuhkan data histori dari server untuk kemudian menampilkannya untuk anda, namun secara sedarhana Git membaca historinya langsung dari basisdata lokal proyek tersebut. Ini berarti anda melihat histori proyek hampir secara instant. Jika anda ingin membandingkan perubahan pada sebuah berkas antara versi saat ini dengan versi sebulan yang lalu, Git dapat mencari berkas yang sama pada sebulan yang lalu dan melakukan pembandingan perubahan secara lokal, bukan dengan cara meminta remote server melakukannya atau meminta server mengirimkan berkas versi yang lebih lama kemudian membandingkannya secara lokal.

Hal ini berarti bahwa sangat sedikit yang tidak bisa anda kerjakan jika anda sedang offline atau berada diluar VPN. Jika anda sedang berada dalam pesawat terbang atau sebuah kereta dan ingin melakukan pekerjaan kecil, anda dapat melakukan commit sampai anda memperoleh koneksi internet hingga anda dapat menguploadnya. Jika anda pulang ke rumah dan VPN client anda tidak bekerja dengan benar, anda tetap dapat bekerja. Pada kebanyakan sistem lainnya, melakukan hal ini cukup sulit atau bahkan tidak mungkin sama sekali. Pada Perforce misalnya, anda tidak dapat berbuat banyak ketika anda tidak terhubung dengan server; pada Subversion dan CVS, anda dapat mengubah berkas, tapi anda tidak dapat melakukan commit pada basisdata anda (karena anda tidak terhubung dengan basisdata). Hal ini mungkin saja bukanlah masalah yang besar, namun anda akan terkejut dengan perbedaan besar yang disebabkannya.

Git Memiliki Integritas

Segala sesuatu pada Git akan melalui proses checksum terlebih dahulu sebelum disimpan yang kemudian direferensikan oleh hasil checksum tersebut. Hal ini berarti tidak mungkin melakukan perubahan terhadap berkas manapun tanpa diketahui oleh Git. Fungsionalitas ini dimiliki oleh Git pada level terendahnya dan ini merupakan bagian tak terpisahkan dari filosofi Git. Anda tidak akan kehilangan informasi atau mendapatkan file yang cacat tanpa diketahui oleh Git.

Mekanisme checksum yang digunakan oleh Git adalah SHA-1 hash. Ini merupakan sebuah susunan string yang terdiri dari 40 karakter heksadesimal (0 hingga 9 dan a hingga f) dan dihitung berdasarkan isi dari sebuah berkas atau struktur direktori pada Git.

14.0.0.1 Secara Umum Git Hanya Menambahkan Data

Ketika anda melakukan operasi pada Git, kebanyakan dari operasi tersebut hanya menambahkan data pada basisdata Git. It is very difficult to get the system to do anything that is not undoable or to make it erase data in any way. Seperti pada berbagai VCS, anda dapat kehilangan atau mengacaukan perubahan yang belum di-commit; namun jika anda melakukan commit pada Git, akan sangat sulit kehilanngannya, terutama jika anda secara teratur melakukan push basisdata anda pada repositori lain.

Hal ini menjadikan Git menyenangkan karena kita dapat berexperimen tanpa kehawatiran untuk mengacaukan proyek. Untuk lebih jelas dan dalam lagi tentang bagaimana Git menyimpan datanya dan bagaimana anda dapat mengembalikan yang hilang, lihat "Under the Covers" pada Bab 9.

14.0.0.2 Tiga Keadaan

Sekarang perhatikan. Ini adalah hal utama yang harus diingat tentang Git jika anda ingin proses belajar anda berjalan lancar. Git memiliki 3 keadaan utama dimana berkas anda dapat berada: committed, modified dan staged. Committed berarti data telah tersimpan secara aman pada basisdata lokal. Modified berarti anda telah melakukan perubahan pada berkas namun anda belum melakukan commit pada basisdata. Staged berarti anda telah menandai berkas yang telah diubah pada versi yang sedang berlangsung untuk kemudian dilakukan commit.

Direktori Git adalah dimana Git menyimpan metadata dan database objek untuk projek anda. Ini adalah bahagian terpenting dari Git, dan inilah yang disalin ketika anda melakukan kloning sebuah repository dari komputer lain.

Direktori kerja adalah sebuah checkout tunggal dari satu versi dari projek. Berkas-berkas ini kemudian ditarik keluar dari basisdata yang terkompresi dalam direktori Git dan disimpan pada disk untuk anda gunakan atau modifikasi.

Staging area adalah sebuah berkas sederhana, umumnya berada dalam direktori Git anda, yang menyimpan informasi mengenai apa yang menjadi commit selanjutnya. Ini terkadang disebut sebagai index, tetapi semakin menjadi standard untuk menyebutnya sebagai staging area.

Alur kerja dasar Git adalah seperti ini:

1. Anda mengubah berkas dalam direktori kerja anda.
2. Anda membawa berkas ke stage, menambahkan snapshotnya ke staging area.
3. Anda melakukan commit, yang mengambil berkas seperti yang ada di staging area dan menyimpan snapshotnya secara permanen ke direktori Git anda.

Jika sebuah versi tertentu dari sebuah berkas telah ada di direktori git, ia dianggap 'committed'. Jika berkas diubah (modified) tetapi sudah ditambahkan ke staging

area, maka itu adalah 'staged'. Dan jika berkas telah diubah sejak terakhir dilakukan checked out tetapi belum ditambahkan ke staging area maka itu adalah 'modified'. Pada Bab 2, anda akan mempelajari lebih lanjut mengenai keadaan-keadaan ini dan bagaimana anda dapat memanfaatkan keadaan-keadaan tersebut ataupun melewatkan bagian 'staged' seluruhnya.

Modifikasi Fungsi yang Ada

Tom melakukan operasi kloning dan menemukan file baru string.c. Dia ingin tahu siapa yang menambahkan file ini ke repositori dan untuk tujuan apa, maka, dia menjalankan perintah git log.

$[tom@CentOS]$ git clone gituser@git.server.com:project.git

Perintah di atas akan menghasilkan hasil sebagai berikut:

Initialized empty Git repository in /home/tom/project/.git/
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
Receiving objects: 100% (6/6), 726 bytes, done.
remote: Total 6 (delta 0), reused 0 (delta 0)

Operasi Clone akan membuat direktori baru di dalam direktori kerja saat ini. Dia mengubah direktori ke direktori yang baru dibuat dan menjalankan perintah git log.

$[tom@CentOS$] cd project/
$[tom@CentOSproject$]git log

Perintah di atas akan menghasilkan hasil sebagai berikut:

commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530
Changed return type of my_strlen to size_t
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit

Setelah mengamati log, dia menyadari bahwa file string.c ditambahkan oleh Jerry untuk mengimplementasikan operasi string dasar. Dia penasaran dengan kode Jerry. Jadi dia membuka string.c di editor teks dan langsung menemukan bug. Dalam fungsi my_strlen, Jerry tidak menggunakan pointer konstan. Jadi, dia memutuskan untuk memodifikasi kode Jerry. Setelah modifikasi, kode tersebut terlihat seperti berikut:

[tom@CentOS project]$ git diff

Perintah di atas akan menghasilkan hasil sebagai berikut:

diff –git a/string.c b/string.c
index 7da2992..32489eb 100644
— a/string.c
+++ b/string.c
@@ -1,8 +1,8 @@
#include <stdio.h>
-size_t my_strlen(char *s)
+size_t my_strlen(const char *s)
{
- char *p = s;
+ const char *p = s;
while (*p)
++p;
}

Setelah melakukan pengujian, dia menyimpan perubahannya.

[$tom@CentOSproject$] git status -s M string.c ?? string
[$tom@CentOSproject$] git add string.c
[$tom@CentOSproject$] git commit -m 'Changed char pointer to const char pointer' [master cea2c00] Changed char pointer to const char pointer 1 files changed, 2 insertions(+), 2 deletions(-)
[$tom@CentOSproject$] git log

perintah diatas akan menghasilkan hasil sebagai berikut:

commit cea2c000f53ba99508c5959e3e12fff493b Author: Tom Cat ¡tom@tutorialspoint.com¿ Date: Wed Sep 11 08:32:07 2013 +0530
Changed char pointer to const char pointer
commit d1e19d316224cddc437e3ed34ec3c931ad803958
Author: Jerry Mouse <jerry@tutorialspoint.com>
Date: Wed Sep 11 08:05:26 2013 +0530
Changed return type of my_strlen to size_t
commit 19ae20683fc460db7d127cf201a1429523b0e319
Author: Tom Cat <tom@tutorialspoint.com>
Date: Wed Sep 11 07:32:56 2013 +0530
Initial commit

Tom menggunakan git push untuk melakukan push atas perubahan yang dilakukannya.

[$tom@CentOSproject$] git push origin master

perintah diatas akan menghasilkan seperti berikut:

Counting objects: 5, done.

Compressing objects: 100 Writing objects: 100 Total 3 (delta 1), reused 0 (delta

0)

To gituser@git.server.com:project.git

d1e19d3..cea2c00 master ¿ master

CHAPTER 15

STASH OPERATION

Salah satu fitur menarik di Git adalah*stashing*. Dengan melakukan*stashing*, developer dapat dengan mudah ' *melenyapkan*' perubahan kode program, melakukan perubahan lain, lalu kembali lagi ke kode program yang sebelumnya dikerjakan. Langkah-langkah tersebut sebenarnya dapat diwakili oleh beberapa perintah Git lainnya, tapi*stashing*membuatnya menjadi mudah karena hanya perlu memanggil satu perintah.

Sebagai contoh, saya akan menggunakan Git yang diakses melalui IntelliJ IDEA. Saya membuat sebuah proyek Groovy baru dengan nama *latihan*. Lalu saya membuat sebuah script Groovy sederhana bernama *Latihan.groovy*untuk menghitung sisa inventory secara FIFO, seperti berikut ini:

```
List pembelian = [10, 20, 30, 40]
List penjualan = [5, 6, 3, 2, 1, 3]
println stokFifo(pembelian, hitungTotal(pembelian, penjualan))
 int hitungTotal(List pembelian, List penjualan) {
 pembelian.sum() - penjualan.sum()}
 List stokFifo(List pembelian, int sisa) {
```

```
 List hasil = []
 pembelian.each { int jumlah -¿
if (sisa > 0) {
int delta = (jumlah >= sisa)? jumlah: sisa // BUG YANG DISENGAJA!
sisa -= delta
hasil << delta
}
}
hasil
}
```

Untuk menambahkan proyek ke dalam repository Git, pilih menu VCS, Import into Version Control,Create Git Repository.

Pada kotak dialog yang muncul, pilih untuk meletakkanrepositorybersamaan dengan lokasi proyek dan kemudian men-klik tombolOK.

Kemudian, klik kanan pada fileLatihan.groovy, memilih menuGit,Add

Sekarang file tersebut telah berada di lokasiindexataustagingdari Git. Berikutnya, commit perubahan. Caranya adalah dengan memilih menuVCS,Commit Changes.

Pilih tombol Commituntuk menyimpan perubahan dalam repository Git lokal. Saat IntelliJ IDEA memunculkan dialog mengenai file lain yang tidak ikut di-commit, pilih No.

Anggap saja ini adalah sebuah aplikasi yang dikembangkan bersama dengan developer lain, yaitu xXx. xXx tersebut telah men- *fetch*repository saya dari remote/upstream, kemudian ia akan men-develop kode program yang berhubungan dengan kode program saya di atas.

Tentu saja saya juga tidak ingin ketinggalan sibuk. Saya menambahkan harga dan perhitungan laba pada kode program saya. saya menulis kode program berikut ini:

```
Map pembelian = [10: 10000, 20: 10100, 30: 10200, 40: 11000]
Map penjualan = [5: 12000, 6: 13000, 3: 11000, 2: 11000, 1: 15000, 3: 12000]
println stokFifo(pembelian, hitungTotal(pembelian.keySet().toList(), pen-
jualan.keySet().toList()))
 int hitungTotal(List pembelian, List penjualan)
pembelian.sum() - penjualan.sum()
 List stokFifo(Map pembelian, int sisa)
List hasil = []
pembelian.each { jumlah, hargaBeli -¿
```

```
if (sisa ¿ 0)
int delta = (jumlah ¿= sisa)? jumlah: sisa // BUG YANG DISENGAJA!
sisa -= delta
hasil ¡¡ [delta, hargaBeli]
}
}
hasil
}
int hitungProfit(List stokFifo, Map penjualan)
```

Saya belum selesai mengetik, bahkan belum sempat menjalankan kode program ketika tiba-tiba telepon berdering. Suara xXx terdengar sangat panik. Dia mengatakan bahwa 30 menit lagi dirinya harus memberikan presentasi ke pengguna, tapi ada yang aneh dengan perhitungan inventory buatan saya! xXx menemukan sebuah kesalahan. Lebih dari itu, xXx meminta saya untuk segera memperbaikinya secepat mungkin sehingga dia bisa men-pull perbaikan dari saya!

Masalahnya: saya sudah mengubah banyak kode program tersebut sehingga tidak sama lagi seperti yang dipakai oleh Lena. Saya tidak ingin menghapus perubahan yang sudah saya buat sejauh ini karena nantinya perubahan ini pasti akan dipakai.

Salah satu solusi yang dapat saya pakai adalah dengan memakai fasilitasstashingdi Git. Saya memilih menuVCS,Git,Stash Changes

Setelah men-klik tombol Create Stash, isi kode program saya secara ajaib kembali lagi seperti semula! Sama persis seperti commit terakhir yang dirujuk oleh HEAD. Saya segera memanfaatkan kesempatan ini untuk mencari dan memperbaiki kesalahan yang ditemukan xXx:

List pembelian = [10, 20, 30, 40]List penjualan = [5, 6, 3, 2, 1, 3]println stokFifo(pembelian, hitungTotal(pembelian, penjualan))int hitungTotal(List pembelian, List penjualan) {pembelian.sum() - penjualan.sum()}List stokFifo(List pembelian, int sisa) {List hasil = []pembelian.each { int jumlah ->if (sisa > 0) {int delta = (jumlah >= sisa)? sisa: jumlahsisa -= deltahasil << delta}}hasil}

Saya segera men-commit perubahan dengan memilih menu**VCS**, **Commit Changes…**. Pada komentar, saya mengisi dengan 'Perbaikan perhitungan sisa inventory yang salah.' dan men-klik tombol **Commit**. Saya kemudian men-*push*perubahan ke *upstream*, sehingga Lena bisa men-*pull*perubahan dari saya. Tidak lama kemudian suara Lena terdengar lega seolah-olah baru saja luput dari malapetaka. Ia memberitahukan bahwa kini semuanya baik-baik saja.

Lalu bagaimana dengan kode program yang sedang saya '*ketik*' sebelum menerima telepon dari Lena? Kemana perginya? Saya bisa mengembalikannya dengan

memilih menu**VCS**,**Git** ,**UnStash Changes...** . Pada kotak dialog yang muncul, saya men-klik tombol**Pop Stash**

Kode program saya akan kembali seperti terakhir kali:
Map pembelian = [10: 10000, 20: 10100, 30: 10200, 40: 11000]Map penjualan = [5: 12000, 6: 13000, 3: 11000, 2: 11000, 1: 15000, 3: 12000]println stokFifo(pembelian, hitungTotal(pembelian.keySet().toList(), penjualan.keySet().toList()))int hitungTotal(List pembelian, List penjualan) {pembelian.sum() - penjualan.sum()}List stokFifo(Map pembelian, int sisa) {List hasil = []pembelian.each { jumlah, hargaBeli ->if (sisa > 0) {int delta = (jumlah >= sisa)? sisa: jumlahsisa -= deltahasil << [delta, hargaBeli]}}hasil}int hitung-Profit(List stokFifo, Map penjualan) {def

Tunggu dulu! Tidak persis sama seperti terakhir kali!! Fasilitas *stashing*bukan saja hanya mengembalikan kode program sebelumnya, tetapi juga melakukan merging dengan perubahan saat ini. Perbaikan yang diminta oleh Lena tidak hilang setelah saya men-pop stash.

Pada contoh ini, hanya terdapat satu file yang berubah. Fasilitas stashing akan semakin berguna bila perubahan telah dilakukan pada banyak file yang berbeda. Tanpa Git dan*stashing*, pada kasus ini, saya harus memakai cara manual yang lebih repot seperti memberikan komentar atau mengedit file untuk sementara.

Membatalkan Apapun

Pada setiap tahapan, Anda mungkin ingin membatalkan sesuatu. Di sini, kita akan membahas beberapa alat dasar untuk membatalkan perubahan yang baru saja Anda lakukan. Harus tetap diingat bahwa kita tidak selalu dapat membatalkan apa yang telah kita batalkan. Ini adalah salah satu area dalam Git yang dapat membuat Anda kehilangan apa yang telah Anda kerjakan jika Anda melakukannya dengan tidak tepat.

Merubah Commit Terakhir Anda

Salah satu pembatalan yang biasa dilakukan adalah ketika kita melakukan commit terlalu cepat dan mungkin terjadi lupa untuk menambah beberapa berkas, atau Anda salah memberikan pesan commit Anda. Jika Anda ingin untuk mengulang commit tersebut, Anda dapat menjalankan commit dengan opsi–ammend:

$ git commit –amend

Perintah ini mangambil area stage Anda dan menggunakannya untuk commit. Jika Anda tidak melakukan perubahan apapun sejak commit terakhir Anda (se-

umpama, Anda menjalankan perintah ini langsung setelah commit Anda sebelumnya), maka snapshot Anda akan sama persis dengan sebelumnya dan yang Anda dapat ubah hanyalah pesan commit Anda.

Pengolah kata akan dijalankan untuk mengedit pesan commit yang telah Anda buat pada commit sebelumnya. Anda dapat ubah pesan commit ini seperti biasa, tetapi pesan commit sebelumnya akan tertimpa.

Sebagai contoh, jika Anda melakukan commit dan menyadari bahwa Anda lupa untuk memasukkan beberapa perubahan dalam sebuah berkas ke area stage dan Anda ingin untuk menambahkan perubahan ini ke dalam commit terakhir, Anda dapat melakukannya sebagai berikut:

$ git commit -m 'initial commit'

$ git add forgotten_file

$ git commit –amend

Ketiga perintah ini tetap akan bekerja di satu commit - commit kedua akan menggantikan hasil dari commit pertama.

Mengeluarkan Berkas dari Area Stage

Dua seksi berikutnya akan menunjukkan bagaimana menangani area stage Anda dan perubahan terhadap direktori kerja Anda. Sisi baiknya adalah perintah yang Anda gunakan untuk menentukan keadaan dari kedua area tersebut juga mengingatkan Anda bagaimana membatalkan perubahannya. Sebagai contoh, mari kita anggap Anda telah merubah dua berkas dan ingin melakukan commit kepada keduanya sebagai dua perubahan terpisah, tetapi Anda secara tidak sengaja mengetikkangit add *dan memasukkan keduanya ke dalam area stage. Bagaimana Anda dapat mengeluarkan salah satu dari keduanya? Perintahgit statusmengingatkan Anda:

$ git add

$ git status

On branch master

Changes to be committed:

(use "git reset HEAD <file>..." to unstage)

#
modified: README.txt

modified: benchmarks.rb

#

Tepat di bawah tulisan ”Changes to be committed”, tercantum anjuran untuk menggunakangit reset HEAD <file>untuk mengeluarkan dari area stage. Mari kita gunakan anjuran tersebut untuk mengeluarkan berkas benchmarks.rb dari area stage:

$ git reset HEAD benchmarks.rb
benchmarks.rb: locally modified

$ git status

On branch master

Changes to be committed:

(use ”git reset HEAD <file>...” to unstage)

#
modified: README.txt

#
Changes not staged for commit:

(use ”git add <file>...” to update what will be committed)

(use ”git checkout – <file>...” to discard changes in working directory)

#
modified: benchmarks.rb

#

Perintahnya terlihat agak aneh, tetapi menyelesaikan masalah. Berkas benchmarks.rb sekarang menjadi terubah dan sudah berada di luar area stage.

Mengembalikan Berkas Terubah

Apa yang terjadi jika Anda menyadari bahwa Anda tidak ingin menyimpan perubahan terhadap berkas benchmarks.rb? Bgaimana kita dapat dengan mudah

mengembalikan berkas tersebut ke keadaan yang sama dengan saat Anda melakukan commit terakhir (atau saat awal menduplikasi, atau bagaimanapun Anda mendapatkannya ketika masuk ke direktori kerja Anda)? Untungnya,git statusmemberitahu Anda lagi bagaimana untuk melakukan hal itu. Pada contoh keluaran sebelumnya, area direktori kerja terlihat seperti berikut:

```
# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout – <file>..." to discard changes in working directory)

#
# modified: benchmarks.rb

#
```

Terlihat secara eksplisit cara Anda dapat membuang perubahan yang telah Anda lakukan (paling tidak, hanya versi Git 1.6.1 atau yang lebih baru yang memperlihatkan cara ini - jika Anda memiliki versi yang lebih tua, kami sangat merekomendasikan untuk memperbaharui Git untuk mendapatkan fitur yang lebih nyaman digunakan). Mari kita lakukan apa yang tertulis di atas:

```
$ git checkout – benchmarks.rb

$ git status

# On branch master

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#
# modified: README.txt

#
```

Anda dapat lihat bahwa perubahan telah dikembalikan. Anda juga seharusnya menyadari bahwa perintah ini juga berbahaya: perubahan apapun yang Anda buat di berkas tersebut akan hilang - Anda baru saja menyalin berkas lain ke perubahan Anda. Jangan pernah gunakan perintah ini kecuali Anda sangat yakin bahwa Anda

tidak menginginkan berkas tersebut. Jika Anda hanya butuh untuk menyingkirkan perubahan untuk sementara, kita dapat bahas tentang penyimpanan (*to stash*) dan pencabangan (*to branch*) di bab berikutnya; kedua cara tersebut secara umum adalah cara yang lebih baik untuk dilakukan.

Ingat bahwa apapun yang dicommit di dalam Git dapat hampir selalu dikembalikan. Bahkan commit yang berada di cabang yang sudah terhapus ataupun commit yang sudah ditimpa dengancommit –amendmasih dapat dikembalikan. Namun, apapun hilang yang belum pernah dicommit besar kumngkinannya tidak dapat dilihat kembali.

15.1 Sistem Kontrol Versi

Version Control System (VCS) adalah perangkat lunak yang membantu pengembang perangkat lunak untuk bekerja sama dan menjaga sejarah lengkap dari pekerjaan mereka.

Di bawah ini adalah fungsi dari VCS a:

- Memungkinkan pengembang untuk bekerja secara bersamaan.
- Tidak memungkinkan Timpa perubahan masing-masing.
- Mempertahankan sejarah setiap versi.

Berikut ini adalah jenis VCS:

- sistem kontrol versi terpusat (CVCS).
- Didistribusikan / Desentralisasi sistem kontrol versi (DVCS).

Dalam bab ini, kita akan berkonsentrasi hanya pada sistem kontrol versi didistribusikan dan terutama pada Git. Git berada di bawah sistem kontrol versi terdistribusi.

15.1.1 Distributed Sistem Kontrol Versi

sistem terpusat kontrol versi (CVCS) menggunakan server pusat untuk menyimpan semua file dan memungkinkan kolaborasi tim. Tapi kelemahan utama dari CVCS adalah titik tunggal kegagalan, yaitu, kegagalan server pusat. Sayangnya, jika server pusat turun selama satu jam, kemudian pada jam itu, tidak ada yang bisa berkolaborasi sama sekali. Dan bahkan dalam kasus terburuk, jika disk server pusat akan rusak dan cadangan yang tepat belum diambil, maka Anda akan kehilangan seluruh sejarah proyek. Di sini, sistem terdistribusi kontrol versi (DVCS) datang ke dalam gambar.

DVCS klien tidak hanya memeriksa snapshot terbaru dari direktori tetapi mereka juga penuh dari repositori tersebut. Jika memutuskan turun, maka repositori dari klien dapat disalin kembali ke server untuk mengembalikannya. Setiap checkout adalah salinan lengkap dari repositori. Git tidak bergantung pada server pusat dan itulah sebabnya Anda dapat melakukan banyak operasi ketika Anda sedang offline. Anda dapat melakukan perubahan, membuat cabang, lihat log, dan melakukan operasi lain ketika Anda sedang offline. Anda memerlukan koneksi jaringan hanya untuk mempublikasikan perubahan dan mengambil perubahan terbaru.

15.1.2 Keuntungan dari Git

free dan open source

Git dirilis di bawah lisensi open source GPL ini. Ini tersedia secara bebas melalui internet. Anda dapat menggunakan Git untuk mengelola proyek kepatutan tanpa membayar satu sen dolar. Karena merupakan open source, Anda dapat mendownload kode sumbernya dan juga melakukan perubahan sesuai dengan kebutuhan Anda.

Cepat dan kecil

Karena sebagian besar operasi dilakukan secara lokal, memberikan manfaat yang sangat besar dalam hal kecepatan. Git tidak bergantung pada server pusat; itu sebabnya, tidak ada kebutuhan untuk berinteraksi dengan server remote untuk setiap operasi. Bagian inti dari Git ditulis dalam C, yang menghindari overhead runtime yang terkait dengan bahasa tingkat tinggi lainnya. Meskipun Git cermin seluruh repositori, ukuran data di sisi client kecil. Ini menggambarkan efisiensi Git di mengompresi dan menyimpan data di sisi client.

backup implisit

Kemungkinan kehilangan data sangat jarang ketika ada beberapa salinan dari itu. Data hadir di setiap sisi klien cermin repositori, karena itu dapat digunakan dalam hal terjadi kecelakaan atau korupsi disk.

Keamanan

Git menggunakan fungsi hash kriptografi umum yang disebut fungsi hash aman (SHA1), untuk nama dan mengidentifikasi objek dalam database. Setiap file dan komit check-dijumlahkan dan diambil oleh checksum-nya pada saat checkout. Ini menyiratkan bahwa, tidak mungkin untuk mengubah file, tanggal, dan pesan komit dan data lainnya dari database Git tanpa mengetahui Git.

Tidak perlu perangkat keras yang kuat

Dalam kasus CVCS, server pusat harus cukup kuat untuk melayani permintaan dari seluruh tim. Untuk tim yang lebih kecil, itu tidak masalah, tetapi sebagai ukuran tim tumbuh, keterbatasan hardware server dapat menjadi hambatan kinerja. Dalam kasus DVCS, pengembang tidak berinteraksi dengan server kecuali mereka butuhkan untuk mendorong atau menarik perubahan. Semua angkat berat terjadi pada sisi klien, sehingga hardware server dapat memang sangat sederhana.

percabangan mudah

CVCS menggunakan mekanisme copy murah, Jika kita membuat cabang baru, itu akan menyalin semua kode ke cabang baru, sehingga memakan waktu dan tidak efisien. Juga, penghapusan dan penggabungan cabang di CVCS rumit dan memakan waktu. Tapi manajemen cabang dengan Git sangat sederhana. Dibutuhkan hanya beberapa detik untuk membuat, menghapus, dan menggabungkan cabang.

CHAPTER 16

MOVE OPERATION

Seperti namanya, operasi memindahkan direktori atau file dari satu lokasi ke lokasi lain. direktori yang dimodifikasi akan muncul sebagai berikut:
[tom@CentOS project] $ pwd

/home/tom/project
[tom@CentOS project] $ ls

README string string.c
[tom@CentOS project] $ mkdir src

[tom@CentOS project] $ git mv string.c src/

[tom@CentOS project] $ git status -s

R string.c −¿ src/string.c
?? string

Untuk membuat perubahan ini permanen, harus mendorong struktur direktori yang dimodifikasi ke repositori jauh sehingga pengembang lain dapat melihat ini.
[tom@CentOS project] $ git commit -m ”Modified directory structure”

[master 7d9ea97] Modified directory structure
1 files changed, 0 insertions(+), 0 deletions(-)
rename string.c =¿ src/string.c (100 %)

[tom@CentOS project] $ git push origin master
Counting objects: 4, done.
Compressing objects: 100 % (2/2), done.
Writing objects: 100 % (3/3), 320 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
e86f062..7d9ea97 master −¿ master

Di gudang lokal Jerry, sebelum operasi penarikan, ia akan menunjukkan struktur direktori lama.

[jerry@CentOS project] $ pwd
/home/jerry/jerry _repo/project

[jerry@CentOS project] $ ls
README string string.c

Tapi setelah operasi tarik, struktur direktori akan diperbarui. Sekarang, Jerry bisa melihat direktori src dan file yang ada di dalam direktori itu.
[jerry@CentOS project] $ git pull
remote: Counting objects: 4, done.
remote: Compressing objects: 100 % (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100 % (3/3), done.
From git.server.com:project
e86f062..7d9ea97 master −¿ origin/master
First, rewinding head to replay your work on top of it...
Fast-forwarded master to 7d9ea97683da90bcdb87c28ec9b4f64160673c8a.

[jerry@CentOS project] $ ls
README src string

[jerry@CentOS project] $ ls src/
string.c

13.1 Hampir Semua Operasi Dilakukan Secara Lokal

Kebanyakan operasi pada Git hanya membutuhkan berkas-berkas dan resource lokal tidak ada informasi yang dibutuhkan dari komputer lain pada jaringan. Jika terbiasa dengan VCS terpusat dimana kebanyakan operasi memiliki overhead latensi jaringan, aspek Git satu ini akan membuat berpikir bahwa para dewa kecepatan telah memberkati Git dengan kekuatan. Karena memiliki seluruh sejarah dari proyek di lokal disk, dengan kebanyakan operasi yang tampak hampir seketika.

Sebagai contoh, untuk melihat history dari proyek, Git tidak membutuhkan data histori dari server untuk kemudian menampilkannya untuk, namun secara sedarhana Git membaca historinya langsung dari basisdata lokal proyek tersebut. Ini berarti melihat histori proyek hampir secara instant. Jika ingin membandingkan perubahan pada sebuah berkas antara versi saat ini dengan versi sebulan yang lalu, Git dapat mencari berkas yang sama pada sebulan yang lalu dan melakukan pembandingan perubahan secara lokal, bukan dengan cara meminta remote server melakukannya atau meminta server mengirimkan berkas versi yang lebih lama kemudian membandingkannya secara lokal.

Hal ini berarti bahwa sangat sedikit yang tidak bisa anda kerjakan jika sedang offline atau berada diluar VPN. Jika sedang berada dalam pesawat terbang atau sebuah kereta dan ingin melakukan pekerjaan kecil, dapat melakukan commit sampai anda memperoleh koneksi internet hingga anda dapat menguploadnya. Jika pulang ke rumah dan VPN client tidak bekerja dengan benar, tetap dapat bekerja. Pada kebanyakan sistem lainnya, melakukan hal ini cukup sulit atau bahkan tidak mungkin sama sekali. Pada Perforce misalnya, tidak dapat berbuat banyak ketika tidak terhubung dengan server; pada Subversion dan CVS, dapat mengubah berkas, tapi tidak dapat melakukan commit pada basisdata (karena tidak terhubung dengan basisdata). Hal ini mungkin saja bukanlah masalah yang besar, namun akan terkejut dengan perbedaan besar yang disebabkannya.

13.2 Segala sesuatu pada Git akan melalui proses checksum terlebih dahulu sebelum disimpan yang kemudian direferensikan oleh hasil checksum tersebut. Hal ini berarti tidak mungkin melakukan perubahan terhadap berkas manapun tanpa diketahui oleh Git. Fungsionalitas ini dimiliki oleh Git pada level terendahnya dan ini merupakan bagian tak terpisahkan dari filosofi Git. Tidak akan kehilangan informasi atau mendapatkan file yang cacat tanpa diketahui oleh Git.

Mekanisme checksum yang digunakan oleh Git adalah SHA-1 hash. Ini merupakan sebuah susunan string yang terdiri dari 40 karakter heksadesimal (0 hingga 9 dan a hingga f) dan dihitung berdasarkan isi dari sebuah berkas atau struktur direktori pada Git. sebuah hash SHA-1 berupa seperti berikut:
24b9da6552252987aa493b52f8696cd6d3b00373

Nilai seperti ini pada berbagai tempat di Git. Faktanya, Git tidak menyimpan nama berkas pada basisdatanya, melainkan nilai hash dari isi berkas.

Ketika melakukan operasi pada Git, kebanyakan dari operasi tersebut hanya menambahkan data pada basisdata Git. Seperti pada berbagai VCS, dapat kehilangan atau mengacaukan perubahan yang belum di-commit; namun jika anda

melakukan commit pada Git akan sangat sulit kehilangannya, terutama jika secara teratur melakukan push basisdata pada repositori lain.

Hal ini menjadikan Git menyenangkan karena kita dapat berexperimen tanpa kehawatiran untuk mengacaukan proyek. Git memiliki 3 keadaan utama dimana berkas anda dapat berada: committed, modified dan staged. Committed berarti data telah tersimpan secara aman pada basisdata lokal. Modified berarti telah melakukan perubahan pada berkas namun belum melakukan commit pada basisdata. Staged berarti telah menandai berkas yang telah diubah pada versi yang sedang berlangsung untuk kemudian dilakukan commit.

Direktori Git adalah dimana Git menyimpan metadata dan database objek untuk projek. Ini adalah bahagian terpenting dari Git, dan inilah yang disalin ketika anda melakukan kloning sebuah repository dari komputer lain.
Direktori kerja adalah sebuah checkout tunggal dari satu versi dari projek. Berkas-berkas ini kemudian ditarik keluar dari basisdata yang terkompresi dalam direktori Git dan disimpan pada disk untuk anda gunakan atau modifikasi.

Staging area adalah sebuah berkas sederhana, umumnya berada dalam direktori Git, yang menyimpan informasi mengenai apa yang menjadi commit selanjutnya. Ini terkadang disebut sebagai index, tetapi semakin menjadi standard untuk menyebutnya sebagai staging area.Alur kerja dasar Git adalah seperti ini:

Jika sebuah versi tertentu dari sebuah berkas telah ada di direktori git dianggap 'committed'. Jika berkas diubah (modified) tetapi sudah ditambahkan ke staging area maka itu adalah 'staged'. Dan jika berkas telah diubah sejak terakhir dilakukan checked out tetapi belum ditambahkan ke staging area maka itu adalah 'modified'.

13.3 Perintah Untuk Membuat Sebuah Proyek

Membuat direktori baru di repositori Git dengan git init. Melakukan direktori setiap saat, benar-benar lokal. Executive git init dalam direktori, Membuat Git repositori. Sebagai contoh, buat item w3big:

```
$ mkdir w3big
$ cd w3big/
$ git init
Initialized empty Git repository in /Users/tianqixin/www/w3big/.git/
# /www/w3big/.git/ Git
```

Sekarang dapat melihat subdirektori git yang dihasilkan dalam proyek. Ini adalah repositori Git, dan semua data yang terkait dengan snapshot dari proyek disimpan di sini.

```
ls -a
.               ..              .git
```

Gunakan git clone repositori Git untuk salinan lokal, sehingga dapat melihat item atau memodifikasinya. Jika membutuhkan sebuah proyek kerjasama dengan orang

lain atau ingin menyalin sebuah proyek, melihat kode, dapat mengkloning proyek. Jalankan:
git clone [url]

Sebagai contoh, kloning proyek pada Github:
$ git clone git@github.com:schacon/simplegit.git
Cloning into 'simplegit'...
remote: Counting objects: 13, done.
remote: Total 13 (delta 0), reused 0 (delta 0), pack-reused 13
Receiving objects: 100 % (13/13), done.
Resolving deltas: 100 % (2/2), done.
Checking connectivity... done.

Setelah kloning selesai di direktori saat ini akan menghasilkan simplegit direktori:
$ Cd simplegit / $ ls README Rakefile lib
operasi akan menyalin semua catatan proyek.

$ ls -a
. .. .git README Rakefile lib
$ cd .git
$ ls
HEAD description info packed-refs
branches hooks logs refs
config index objects
Secara default, Git akan mengikuti nama URL yang tersedia item untuk

membuat direktori proyek lokalditunjukkan. URL biasanya nama item terakhir / setelah. Jika ingin nama yang berbeda dapat menambahkan nama yang inginkan setelah perintah.

13.4 Snapshot Dasar

Pekerjaan Git adalah untuk membuat dan menyimpan snapshot dari proyek dan setelah snapshot dan membandingkan. Bab ini akan tentang menciptakan sebuah snapshot dari proyek dan mengirimkan pengenalan perintah.

git add perintah untuk menambahkan file ke cache, seperti yang tambahkan dua file berikut:
$ touch README
$ touch hello.php
$ ls
README hello.php

$ git status -s
?? README
?? hello.php
$
Perintah git status digunakan untuk melihat status proyek. Selanjutnya jalankan git add perintah untuk menambahkan file:
$ git add README hello.php
Sekarang jalankan git status, dapat melihat dua dokumen tersebut telah ditambahkan

untuk pergi.
$ git status -s
A README
A hello.php
$

Proyek baru, menambahkan semua file yang sama, kita dapat menggunakangit add. Perintah untuk menambahkan semua file dalam proyek saat ini. Sekarang memodifikasi file README:
$ vim README
¡pre¿
¡p¿ README Ł¡b¿ # w3big Git ¡/b¿¡/p¿
¡p¿ git status¡/p¿
$ git status -s
AM README
A hello.php

”AM” status berarti bahwa file tersebut setelah kami menambahkannya ke cache ada perubahan. Setelah perubahan menjalankan git add perintah untuk menambahkannya ke cache:
$ git add .
$ git status -s
A README
A hello.php
Bila ingin perubahan yang terkandung dalam snapshot laporan yang akan datang dalam waktu, harus menjalankan git add.

Git status untuk melihat setelah komit terakhir jika ada perubahan. Menunjukkan perintah ini ketika ditambahkan -s parameter untuk mendapatkan hasil yang singkat. Jika tidak menambahkan parameter ini akan keluaran rinci:
$ git status
On branch master
Initial commit

Changes to be committed:

(use ”git rm –cached ¡file¿...” to unstage)

new file: README

new file: hello.php

Status git diff git eksekutif untuk melihat rincian hasil eksekusi. Git perintah diff dan menampilkan cache write telah dimodifikasi tapi belum ditulis ke cache perubahan perbedaan. git diff Ada dua skenario utama.

- Perubahan tidak cache:diff git
- Lihatperubahan cache: git diff –cached
- Lihat cache dan uncached semuaperubahan: git diff KEPALA
- Tampilkan ringkasan daripada seluruhdiff: git diff –stat

Masukkan berikut dalam file hello.php:
¡?php
echo 'www.w3big.com';
?¿
$ git status -s
A README
AM hello.php
$ git diff
diff –git a/hello.php b/hello.php
index e69de29..69b5711 100644
— a/hello.php
+++ b/hello.php
@@ -0,0 +1,3 @@
+¡?php
+echo 'www.w3big.com';
+?¿
Menampilkan status git pada untuk berubah setelah update atau menulis garis perubahan cache dengan garis dan git diff menunjukkan secara spesifik apa perubahan tersebut. Selanjutnya melihat git berikutnya diff pelaksanaan –cached hasil:
$ git add hello.php
$ git status -s
A README
A hello.php
$ git diff –cached
diff –git a/README b/README
new file mode 100644
index 0000000..8f87495
— /dev/null
+++ b/README
@@ -0,0 +1 @@

```
+ # w3big Git
diff –git a/hello.php b/hello.php
new file mode 100644
index 0000000..69b5711
— /dev/null
+++ b/hello.php
@@ -0,0 +1,3 @@
+¡?php
+echo 'www.w3big.com';
+?¿
```

Gunakan git menambahkan perintah ingin menulis isi dari buffer snapshot, dan mengeksekusi git commit akan menambahkan konten ke gudang penyangga. Git mengirimkan masing-masing nama dan alamat e-mail yang tercatat, sehingga langkah pertama perlu mengkonfigurasi nama pengguna dan alamat e-mail.

```
$ git config –global user.name 'w3big'
$ git config –global user.email test@w3big.com
```

Berikutnya menulis caching, dan menyerahkan semua perubahan

hello.php tersebut. Dalam contoh pertama menggunakan opsi -m untuk memberikan baris perintah untuk mengirimkan komentar.

```
$ git add hello.php
$ git status -s
A  README
A  hello.php
$ $ git commit -m ''
[master (root-commit) d32cf1f]
2 files changed, 4 insertions(+)
create mode 100644 README
create mode 100644 hello.php
```

Sekarang telah mencatat snapshot. Jika kita jalankan git status:

```
$ git status
# On branch master
nothing to commit (working directory clean)
```

Output di atas menunjukkan bahwa setelah pengajuan terakhir, tidak membuat perubahan apapu. Jika tidak menetapkan opsi -m, Git mencoba untuk membuka editor untuk mengisi informasi yang disampaikan. Git jika tidak dapat menemukan informasi yang relevan dalam konfigurasi, default akan membuka vim. Layar akan terlihat seperti ini:

```
# Please enter the commit message for your changes. Lines starting
# with ' #' will be ignored, and an empty message aborts the commit.
```

```
# On branch master
# Changes to be committed:
#   (use "git reset HEAD ¡file¿..." to unstage)
#
# modified:  hello.php
#
~
~
".git/COMMIT_EDITMSG" 9L, 257C
```

Jika berpikir git add disampaikan proses cache yang terlalu rumit, Git juga memungkinkan Anda untuk menggunakan opsi -a untuk melewatkan langkah ini. format perintah adalah sebagai berikut:
git commit -a

Mari memodifikasi file hello.php sebagai berikut:

```
¡?php
echo 'www.w3big.com';
echo 'www.w3big.com';
?¿
```

Kemudian jalankan perintah berikut:

```
git commit -am ' hello.php '
[master 71ee2cb]  hello.php
1 file changed, 1 insertion(+)
```

Git reset perintah HEAD untuk menghapus konten cache. Mari mengubah file berkas README, sebagai berikut:

```
# w3big Git
#
```

File hello.php diubah sebagai berikut:

```
¡?php
echo 'www.w3big.com';
echo 'www.w3big.com';
echo 'www.w3big.com';
?¿
```

Sekarang setelah dua file diubah disampaikan ke zona penyangga, sekarang ingin membatalkan salah satu dari cache, sebagai berikut:

```
$ git status -s
M README
M hello.php
$ git add .
$ git status -s
```

```
M  README
M  hello.pp
$ git reset HEAD – hello.php
Unstaged changes after reset:
M                hello.php
$ git status -s
M  README
M hello.php
```

Sekarang menjalankan git commit, perubahan hanya akan diserahkan berkas README, tapi hello.php tidak.

```
$ git commit -m ''
[master f50cfda]
1 file changed, 1 insertion(+)
$ git status -s
M hello.php
```

Melihat file perubahan hello.php dan untuk pengajuan. Maka dapat menggunakan perintah berikut untuk memodifikasi hello.php menyerahkan:

```
$ git commit -am ' hello.php '
[master 760f74d]  hello.php
1 file changed, 1 insertion(+)
$ git status
On branch master
nothing to commit, working directory clean
```

Singkatnya, melakukan git reset HEAD untuk membatalkan sebelum git add untuk menambahkan, tetapi tidak ingin untuk memasukkan dalam cache snapshot di commit selanjutnya.

Entri rm git akan dihapus dari cache. ulang KEPALA git ini membatalkan entri cache yang berbeda. ”Batal Cache”, yang berarti bahwa pemulihan akan membuat perubahan ke cache. Secara default, git file rm akan dihapus dari file cache dan hard drive (direktori kerja). Jika ingin menyimpan file dalam direktori kerja, dapat menggunakangit rm –cached: Seperti kita menghapus hello.php file:

```
$ git rm hello.php
rm 'hello.php'
$ ls
README
```

Tidak menghapus file dari ruang kerja:

```
$ git rm –cached README
rm 'README'
$ ls
README
```

Git perintah mv untuk melakukan semua hal yanggit rm perintah operasi –cached,mengubah nama file pada disk, dan kemudian jalankan git add untuk menambahkan file baru ke cache. README pertama kita hapus hanya menambahkan kembali:

```
$ git add README
```

Kemudian nama yang sama yaitu:

```
$ git mv README README.md
$ ls
README.md
```

CHAPTER 17

RENAME OPERATION

Sampai sekarang, baik Tom dan Jerry menggunakan perintah manual untuk menyusun proyek mereka. Sekarang, Jerry memutuskan untuk membuat Makefile untuk proyek mereka dan juga memberi nama yang tepat untuk file ”string.c”.
[jerry@CentOS project] $ pwd
/home/jerry/jerry _repo/project
[jerry@CentOS project] $ ls

README src
[jerry@CentOS project] $ cd src/

[jerry@CentOS src] $ git add Makefile

[jerry@CentOS src] $ git mv string.c string _operations.c

[jerry@CentOS src] $ git status -s

A Makefile
R string.c −¿ string _operations.c

Git menunjukkan R sebelum nama file untuk menunjukkan bahwa file telah diganti namanya.

Untuk komit operasi, Jerry menggunakan - bendera, yang membuat git komit secara otomatis mendeteksi file yang dimodifikasi.
[jerry@CentOS src] $ git commit -a -m 'Added Makefile and renamed strings.c to string _operations.c '

[master 94f7b26] Added Makefile and renamed strings.c to string _operations.c
1 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 src/Makefile
rename src/ {string.c =¿ string _operations.c } (100 %)

Setelah komit, dia mendorong perubahannya ke repositori.
[jerry@CentOS src] $ git push origin master

Perintah di atas akan menghasilkan hasil sebagai berikut:
Counting objects: 6, done.
Compressing objects: 100 % (3/3), done.
Writing objects: 100 % (4/4), 396 bytes, done.
Total 4 (delta 0), reused 0 (delta 0)
To gituser@git.server.com:project.git
7d9ea97..94f7b26 master −¿ master

Sekarang, pengembang lain dapat melihat modifikasi ini dengan memperbarui repositori lokal mereka.

Kegunaan utama dari sistem kontrol versi ialah sebagai alat untuk manajemen kode program. Terdapat dua kegunaan utama dari sistem ini, yaitu:
Menggabungkan perubahan-perubahan kode dari versi lama (misal: untuk mengembalikan fitur yang telah dihapus) ataupun menggabungkan perubahan dari orang lain (misal: menggabungkan fitur yang dikembangkan oleh anggota tim lain).

14.1 Intalasi Git

git berjalan pada semua sistem operasi populer (Mac, Windows, Linux). Jika menggunakan Windows atau Mac, masuk ke situs utama git pada lalu lakukan download dan instalasi software tersebut. Pengguna Linux dapat melakukan instalasi melalui repositori distribusi yang dilakukan, melalui perintah sejenis:
yum install git

pada repositori berbasis RPM, atau perintah
apt-get install git

Untuk repositori berbasis deb. Kembali lagi, perintah hanya diberikan untuk distribusi paling populer (Debian / Ubuntu dan RedHat / Fedora), karena keterbatasan ruang. Jika menggunakan distrusi lain (seperti Gentoo atau Arch, maka diasumsikan telah mengetahui cara instalasi git atau perangkat lunak lain pada umumnya).

Khusus untuk sistem operasi Windows, pastikan instalasi anda diambil dari , karena pada paket yang tersedia di website tersebut telah diikutkan juga OpenSSH, yang akan sangat berguna jika ingin berkolaborasi dengan programmer lain. Perintah git juga harus memberikan respon yang benar:

```
bert@LYNNSLENIA ~
$ git
usage: git [–version] [–exec-path[=¡path¿]] [–html-path] [–man-path] [–info-path]
       [-p |–paginate |–no-pager] [–no-replace-objects] [–bare]
       [–git-dir=¡path¿] [–work-tree=¡path¿] [–namespace=¡name¿]
       [-c name=value] [–help]
       ¡command¿ [¡args¿]
The most commonly used git commands are:
  add       Add file contents to the index
  bisect    Find by binary search the change that introduced a bug
  branch    List, create, or delete branches
  checkout  Checkout a branch or paths to the working tree
  clone     Clone a repository into a new directory
  commit    Record changes to the repository
 diff      Show changes between commits, commit and working tree, etc
  fetch     Download objects and refs from another repository
  grep      Print lines matching a pattern
  init      Create an empty git repository or reinitialize an existing one
  log       Show commit logs
  merge     Join two or more development histories together
  mv        Move or rename a file, a directory, or a symlink
  pull      Fetch from and merge with another repository or a local branch
  push      Update remote refs along with associated objects
  rebase    Forward-port local commits to the updated upstream head
  reset     Reset current HEAD to the specified state
  rm        Remove files from the working tree and from the index
  show      Show various types of objects
  status    Show the working tree status
  tag       Create, list, delete or verify a tag object signed with GPG
 See 'git help ¡command¿' for more information on a specific command.

bert@LYNNSLENIA ~
$
```

14.2 Inisiasi

Untuk dapat menggunakan sistem kontrol versi, terlebih dahulu kita harus mempersiapkan repositori. Sebuah repositori menyimpan seluruh versi dari kode program kita. Tidak usah takut, karena repositori tidak akan memakan banyak ruang *hard disk*, karena penyimpanan tidak dilakukan terhadap keseluruhan file. Repositori hanya akan menyimpan *perubahan* yang terjadi pada kode kita dari satu versi ke

versi lainnya. Bahasa kerennya, repositori hanya menyimpan delta dari kode pada setiap versinya.

Pada (di saat kontrol versi yang populer adalah cvs dan programmer pada umumnya berjanggut putih), membangun repositori kode baru adalah hal yang sangat sulit dilakukan. Harus memiliki sebuah *server* khusus yang dapat diakses oleh seluruh anggota tim. Jika server tidak dapat diakses karena jaringan rusak atau internet putus, maka tidak dapat melakukan kontrol versi (dan harus kembali ke metode direktori, atau tidak bekerja).

git merupakan sistem kontrol versi terdistribusi, yang berarti git dapat dijalankan tanpa perlu adanya repositori terpusat. Yang diperlukan untuk membuat repositori ialah mengetikkan perintah tertentu di direktori utama. Mulai membuat repositori baru.:

```
bert@LYNNSLENIA ~
$ cd Desktop/projects/git-tutor/
bert@LYNNSLENIA ~/Desktop/projects/git-tutor
$ ls
bert@LYNNSLENIA ~/Desktop/projects/git-tutor
$
```

Menambahkan kode baru ke dalam direktori ini. Buat sebuah file baru yang bernama

cerita.txt di dalam direktori tersebut:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor
$ echo ”ini adalah sebuah cerita” ¿ cerita.txt
 bert@LYNNSLENIA ~/Desktop/projects/git-tutor

$ ls
cerita.txt
```

kemudian masukkan perintah git init untuk melakukan inisialisasi repositori:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor
$ git init
Initialized empty Git repository in c:/Users/bert/Desktop/projects/git-tutor/.git/
```

Setelah melakukan inisialisasi, git secara otomatis akan membuat direktori .git pada repositori (lihat potongan kode di bawah). Direktori tersebut merupakan direktori yang digunakan oleh git untuk menyimpan basis data delta kode, dan berbagai metadata lainnya. Mengubah direktori tersebut dapat menyebabkan hilangnya seluruh *history* dari kode.

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ ls -a
. .. .git cerita.txt
```

14.3 Penambahan File ke Repository

Penyimpanan sejarah dapat dimulai dari saat pertama: kapan file tersebut dibuat dan ditambahkan ke dalam repositori. Untuk menambahkan file ke dalam repositori, gunakan perintah git add:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git add .
warning: LF will be replaced by CRLF in cerita.txt.
The file will have its original line endings in your working directory.
```

Secara sederhana, sintaks dari perintah git add adalah sebagai berikut:

```
git add [nama file atau pola]
```

Memasukkan nama file dalam perintah git add pada dasarnya akan memerintahkan git untuk menambahkan **semua** file baru dalam repositori. Jika hanya ingin menambahkan satu file (misalkan ada file yang belum yakin akan ditambahkan ke repositori), nama file spesifik dapat dimasukkan:

```
git add cerita.txt
```

Setelah menambahkan file ke dalam repositori, harus melakukan *commit*. Perintah *commit* memberitahukan kepada git untuk menyimpan sejarah dari file yang telah ditambahkan. Pada git, penambahan, perubahan, ataupun penghapusan sebuah file baru akan tercatat jika perntah *commit* telah dijalankan. Mari lakukan *commit* dengan menjalankan perintah git commit:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git commit
```

Jika langkah di atas diikuti dengan benar, maka kembali ke git bash, dengan pesan berikut:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git commit
[master (root-commit) 1d4cdc9] Inisialisasi repo. Penambahan cerita.txt.
warning: LF will be replaced by CRLF in cerita.txt.
The file will have its original line endings in your working directory.
1 file changed, 1 insertion(+)
create mode 100644 cerita.txt
```

14.4 Mengubah Isi File

Kegunaan utama kontrol versi (yang tercermin dari namanya) ialah melakukan manajemen perubahan secara otomatis untuk kita. dan kemudian jalankan perintah git commit lagi:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git commit
# On branch master
# Changes not staged for commit:
```

```
#   (use "git add ¡file¿..." to update what will be committed)
#   (use "git checkout – ¡file¿..." to discard changes in working directory)
#
#       modified:  cerita.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Perhatikan bahwa git secara otomatis mengetahui file mana saja yang berubah, tetapi tidak melakukan pencatatan perubahan tersebut. Untuk memerintahkan git mencatat perubahan tersebut, gunakan perintah git commit -a:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git commit -a
[master 61c4707] Kapitalisasi dan melengkapi kalimat.
1 file changed, 1 insertion(+), 1 deletion(-)
```

Selain melakukan perubahan, tentunya terkadang kita ingin mengetahui perubahan-perubahan apa saja yang terjadi selama pengembangan. Untuk melihat daftar perubahan yang telah dilakukan, kita dapat menggunakan perintah git log:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git log
commit 61c47074ee583dbdd16fa9568019e80d864fb403
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date:   Sun Dec 23 16:36:46 2012 +0700
    Kapitalisasi dan melengkapi kalimat.

commit 1d4cdc9350570230d352ef19aededf06769b0698

Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date:   Sun Dec 23 16:10:33 2012 +0700
    Inisialisasi repo. Penambahan cerita.txt.
```

Mari jalankan perintah git log sekali lagi, untuk melihat hasil pekerjaan kita sejauh ini:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git log
commit 28dabb1c54a086cce567ecb890b10339416bcbfa
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date:   Sun Dec 23 16:49:21 2012 +0700
    Penambahan misteri terbesar di dunia.

commit 61c47074ee583dbdd16fa9568019e80d864fb403

Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date:   Sun Dec 23 16:36:46 2012 +0700
    Kapitalisasi dan melengkapi kalimat.
```

commit 1d4cdc9350570230d352ef19aededf06769b0698

Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date: Sun Dec 23 16:10:33 2012 +0700
Inisialisasi repo. Penambahan cerita.txt.

git memungkinkan kita untuk mengembalikan kode ke dalam keadaan sebelumnya, yaitu *commit* terakhir. Melakukan pengembalian kode ini dengan menggunakan perintah git checkout seperti berikut:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git checkout HEAD – cerita.txt
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ ls
cerita.txt
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ cat cerita.txt
Ini adalah sebuah cerita tentang seekor kera yang terkurung dan terpenjara dalam goa.
Kera ini bernama Sun Go Kong. Dari manakah Sun Go Kong berasal?
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
```

Parameter HEAD pada perintah yang kita jalankan merupakan parameter untuk memberitahukan git checkout bahwa kita ingin mengembalikan kode pada revisi terakhir (HEAD dalam istilah git). Karena hanya ingin mengembalikan file cerita.txt, maka kita harus memberitahukan git checkout, melalui parameter – cerita.txt. Perintah git checkout juga memiliki banyak kegunaan lainnya selain mengembalikan kode ke revisi tertentu.

Untuk melihat bagaimana fitur ini bekerja, mari lakukan perubahan pada repositori terlebih dahulu. Tambahkan sebuah file baru ke dalam repositori:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ ls
cerita.txt
 bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)

$ echo ”Seekor kera, terpuruk, terpenjara dalam goa. Di gunung suci sunyi
tempat hukuman para dewa.” ¿ lagu-intro.txt
 bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)

$ ls
cerita.txt  lagu-intro.txt
 bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)

$ git add .
warning: LF will be replaced by CRLF in lagu-intro.txt.
The file will have its original line endings in your working directory.
```

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)

$ git commit
[master 03d0628] Penambahan lagu intro.
warning: LF will be replaced by CRLF in lagu-intro.txt.
The file will have its original line endings in your working directory.
1 file changed, 1 insertion(+)
create mode 100644 lagu-intro.txt
```

Kemudian kita akan melakukan edit terhadap cerita.txt dan mengganti nama lagu-intro.txt menjadi lagu-intro-awal.txt:

```
ert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ ls
cerita.txt  lagu-intro.txt
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ notepad cerita.txt
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ mv lagu-intro.txt lagu-intro-awal.txt
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ ls
cerita.txt  lagu-intro-awal.txt
```

Setelah melakukan perubahan tersebut, kita mengalami amnesia sesaat karena kucing kantor jatuh ke kepala kita (kucing yang menyebalkan!). Karena telah lupa akan perubahan yang dilakukan, kita dapat melihat apa saja yang berubah dengan menggunakan perintah git status:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git status
# On branch master
# Changes not staged for commit:
#   (use ”git add/rm ¡file¿...” to update what will be committed)
#   (use ”git checkout – ¡file¿...” to discard changes in working directory)
#
#       modified:  cerita.txt
#       deleted:   lagu-intro.txt
#
# Untracked files:
#   (use ”git add ¡file¿...” to include in what will be committed)
#
#       lagu-intro-awal.txt
no changes added to commit (use ”git add” and/or ”git commit -a”)
```

Perhatikan bahwa terdapat dua bagian dari status yang diberikan:
”Changes not staged for commit ” menampilkan daftar file yang berubah, tetapi belum di-*commit*. File yang tercatat ini termasuk file yang diubah dan dihapus.

”Untracked files ” menampilkan file yang belum ditambahkan ke dalam repositori. Jika ingin melihat apa saja yang diubah pada file cerita.txt, kita dapat menggunakan perintah git diff:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git diff cerita.txt
diff –git a/cerita.txt b/cerita.txt
index 846114d..dbcb596 100644
— a/cerita.txt
+++ b/cerita.txt
@@ -1,3 +1,3 @@
Ini adalah sebuah cerita tentang seekor kera yang terkurung dan terpenjara dala
 -Kera ini bernama Sun Go Kong. Dari manakah Sun Go Kong berasal?

+Kera ini bernama Sun Go Kong. Dari manakah Sun Go Kong berasal???!
(END)
```

Format yang ditampilkan mungkin agak membingungkan, tetapi tidak usah takut, karena bagian yang perlu diperhatikan hanyalah pada bagian yang bertanda - dan +. Pada git bash, bahkan bagian ini diberi warna (merah untuk - dan hijau untuk +). Tanda +, tentunya berarti bagian yang ditambahkan, dan tanda - berarti bagian yang dihapus. Dengan melihat perubahan pada baris yang bersangkutan, kita dapat mengetahui bahwa ? diubah menjadi ???! pada akhir baris.

Setelah mengetahui perubahan yang dilakukan, dan menganggap perubahan tersebut aman untuk di-*commit*, kita lalu dapat melakukan *commit* seperti biasa:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git add lagu-intro-awal.txt
warning: LF will be replaced by CRLF in lagu-intro-awal.txt.
The file will have its original line endings in your working directory.

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git commit
[master 306f422] Dramatisasi cerita dan perubahan nama file lagu.
warning: LF will be replaced by CRLF in lagu-intro-awal.txt.
The file will have its original line endings in your working directory.
1 file changed, 1 insertion(+)
create mode 100644 lagu-intro-awal.txt

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git log
commit 306f42258f4bfee95d10396777391ae013bc6edd
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date:   Sun Dec 23 18:22:30 2012 +0700

    Dramatisasi cerita dan perubahan nama file lagu.

commit 03d06284462f7fc43b610d522678f4f22cdd9a40
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
```

Date: Sun Dec 23 18:08:10 2012 +0700

Penambahan lagu intro.

commit 28dabb1c54a086cce567ecb890b10339416bcbfa
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date: Sun Dec 23 16:49:21 2012 +0700

Penambahan misteri terbesar di dunia.

commit 61c47074ee583dbdd16fa9568019e80d864fb403
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date: Sun Dec 23 16:36:46 2012 +0700

Kapitalisasi dan melengkapi kalimat.

commit 1d4cdc9350570230d352ef19aededf06769b0698
Author: Alex Xandra Albert Sim ¡bertzzie@gmail.com¿
Date: Sun Dec 23 16:10:33 2012 +0700

Inisialisasi repo. Penambahan cerita.txt.

14.5 Membaca File Lama, dan Menjalankan Mesin Waktu

Nomor revisi, seperti yang telah dijelaskan sebelumnya, berguna sebagai tanda untuk memisahkan antara satu *commit* dengan *commit* lainnya. Misalnya jika ingin melihat isi file cerita.txt pada saat awal pertama kali dibuat, kita dapat menggunakan perintah git show, yang sintaksnya adalah:

git show [nomor revisi]:[nama file]

contoh pengunaan:

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git show 1d4cdc:cerita.txt
ini adalah sebuah cerita
```

Perhatikan bahwa nomor commit yang dimasukkan hanyalah enam karakter saja. Jika keenam karakter tersebut sama untuk beberapa nomor *commit*, kita baru perlu memasukkan karakter selanjutnya, sampai tidak terdapat konflik nama lagi.

Sesuai dengan nomor revisi dengan menggunakan git checkcout yang telah dijelaskan sebelumnya. Contohnya :

```
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ ls
cerita.txt  lagu-intro-awal.txt

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
```

```
$ cat cerita.txt
Ini adalah sebuah cerita tentang seekor kera yang terkurung dan terpenjara dalam
goa.

Kera ini bernama Sun Go Kong. Dari manakah Sun Go Kong berasal???!

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git checkout 61c470 cerita.txt

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ cat cerita.txt
Ini adalah sebuah cerita tentang seekor kera yang terkurung dan terpenjara dalam
goa.
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git checkout 1d4cdc cerita.txt

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ cat cerita.txt
ini adalah sebuah cerita

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git checkout 03d0628 cerita.txt

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ cat cerita.txt
Ini adalah sebuah cerita tentang seekor kera yang terkurung dan terpenjara dalam
goa.

Kera ini bernama Sun Go Kong. Dari manakah Sun Go Kong berasal?
bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ git checkout HEAD cerita.txt

bert@LYNNSLENIA ~/Desktop/projects/git-tutor (master)
$ cat cerita.txt
Ini adalah sebuah cerita tentang seekor kera yang terkurung dan terpenjara dalam
goa.

Kera ini bernama Sun Go Kong. Dari manakah Sun Go Kong berasal???!
```

Perhatikan bahwa pada saat menggunakan perintah git checkout, menggunakan cat untuk melihat isi file. Hal ini dikarenakan git checkout benar-benar mengubah file yang ada pada repositori, berbeda dengan git show yang hanya menampilkan file tersebut pada revisi tertentu.

REFERENCES

[Kil76] J. S. Kilby, "Invention of the Integrated Circuit," *IEEE Trans. Electron Devices,* **ED-23,** 648 (1976).

[Ham62] R. W. Hamming, *Numerical Methods for Scientists and Engineers*, Chapter N-1, McGraw-Hill, New York, 1962.

[Hu86] J. Lee, K. Mayaram, and C. Hu, "A Theoretical Study of Gate/Drain Offset in LDD MOSFETs" *IEEE Electron Device Lett.,* **EDL-7**(3). 152 (1986).

[Ber87] A. Berenbaum, B. W. Colbry, D.R. Ditzel, R. D Freeman, and K.J. O'Connor, "A Pipelined 32b Microprocessor with 13 kb of Cache Memory," it Int. Solid State Circuit Conf., Dig. Tech. Pap., p. 34 (1987).