<https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_manually_verifying_the_change>

Gerrit Code Review - A Quick Introduction

Version V2.14.5.1

Table of Contents

* [What is Gerrit?](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_what_is_gerrit)
* [Where does Gerrit fit in?](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_where_does_gerrit_fit_in)
* [The Life and Times of a Change](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_the_life_and_times_of_a_change)
  + [Cloning the Repository](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_cloning_the_repository)
  + [Creating the Review](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_creating_the_review)
  + [Reviewing the Change](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_reviewing_the_change)
  + [Reworking the Change](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_reworking_the_change)
  + [Trying out the Change](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_trying_out_the_change)
  + [Manually Verifying the Change](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_manually_verifying_the_change)
  + [Submitting the Change](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/intro-quick.html#_submitting_the_change)

Gerritはgit上で動作するウェブベースのコードレビューツールです。しかし、あなたがこのガイドを読んでいることから察するに、恐らくそんなことは既にご存じなのでしょう。このイントロダクションの目的は、皆さんの質問に答えることです。私にとって適切なツールなのですか？これは私の仕事や組織にフィットしますか？といった質問に。

What is Gerrit?

これを読んでいるあなた方は、恐らく一般的なコードレビューツールの利点については納得しているけれども、レビュー作業を簡単にするために、何かしらの技術サポートを必要としている人々だと思われます。

コードレビューの方法は、人によって異なります。ある人は会議でプロジェクターを使い、チーム全員でコードを一行ずつ見ていくのがコードレビューだと言うでしょう。またある人は、コードをちらっと見ただけでコミットするのがレビューだと言うかもしれません。

Gerritは全てのコミットに対して、事前にレビューするための軽いフレームワークを提供するように設計されています。コードの変更内容はGerritにアップロードされた後、レビューが完了するまでは、リポジトリには反映されない仕組みです。

色々な意味で、これは単純にオープンソースソフトウェアにおいてパッチを提出する時のプロセスをサポートするツールだと言えます。オープンソースソフトウェアでは、パッチはプロジェクトに取り込まれる前にプロジェクトメンバーの方々にレビューされます。

しかし、Gerrirではレビュープロセスの単純化を更に進めて、全てのコミッター（コミットする人）にとってよりシンプルに、変更内容が適用される前に必ずレビューされることを保証します。

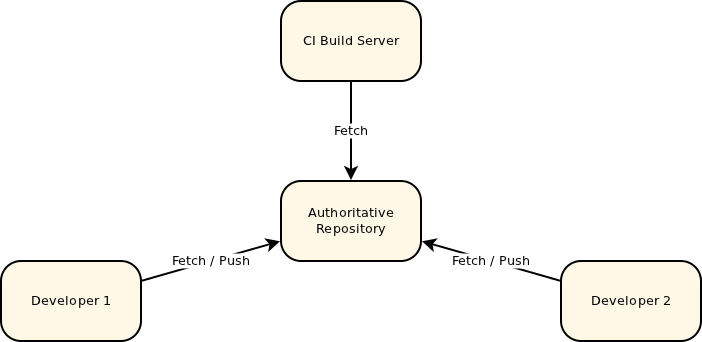
そのためGerritは全てのユーザーが信用されたコミッターである場合、例えば非オープンソースソフトウェアを使った商業的開発の開発環境でも同様に利用価値があります。どちらにしても、コードの品質向上と保守性の観点から、コードはレビューされるのが望ましいと言えます。もしも、たった一人しかそのコードを知らないとすると、その人が会社を去った後はコードを保守するのが難しくなるのです。

簡単に言うと、Gerritはステージングエリアです。変更内容はコミットされる前にそこでレビューされる仕組みです。Gerritはまた、レビュープロセスの実装でもあります。変更内容に対するノートやコメントを記録して、変更内容に対する議論の場を提供します。一般的にこのような仕組みは分散チームにおいて有効です。そのようなチームでは、変更内容について直接、面と向かって(face-to-face)会話することは不可能です。同じ場所で働いているチームであったとしても、レビューツールを使うことには価値があります。なぜなら、レビューワーは、自分の都合の良い時にレビューすることが出来るからです。レビューツールを使うことで、開発者は変更内容について集中した状態で、レビューを開き、変更内容について説明することが出来ます。このようなツールがないと、開発者はコードレビューをしてもらうために他の開発者の作業を中断させたり、変更内容の説明のために何度も頭を切り替えたりする必要があります。

またレビューツールを使うと会話の記録が残るので、「何のために変更するのか」という、避けられない質問に答えるのに便利です。

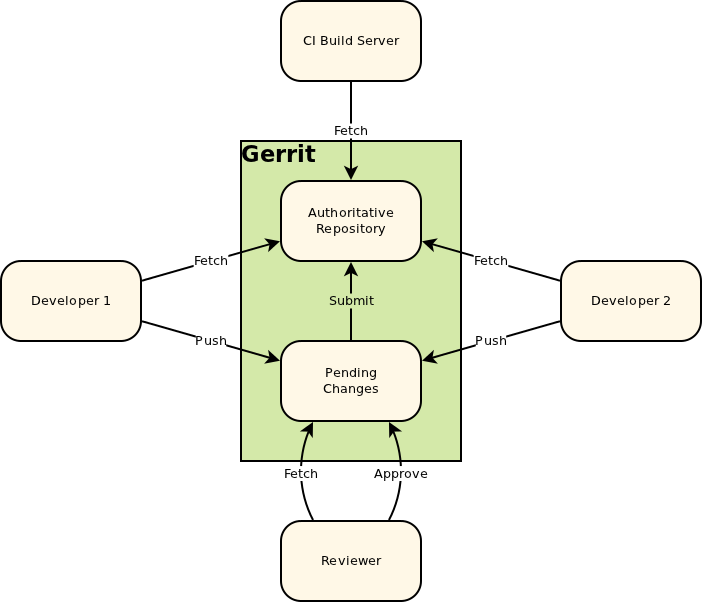
Where does Gerrit fit in?

どんなチームであれ、２人以上のメンバーがいるのであれば、何かしら集中管理リポジトリを使っています（使うべきです）。Gitは理論的には集中管理リポジトリがなくても運用出来ますが、実際には、通常は集中管理リポジトリを立てて運用されています。これはプロジェクトの正式なコピーを提供します。このリポジトリを通して、開発者はfetchやpushを行ったり、ビルドサーバーや他のツールがソースコードを拾ったりします。



*Figure 1. Central Source Repository*

Gerritは集中リポジトリに配置されて、pendingされている変更内容を記録する、という追加機能を提供します。Fetchは集中管理リポジトリから行えますが、pushをする時には、変更内容をpending changes locationに対してpushします。変更内容はレビューを実施して、レビュー完了をしない限り、集中管理リポジトリには反映されません。



*Figure 2. Gerrit in place of Central Repository*

他のリポジトリホスティングサービスと同様に、Gerritも強力な [access control model](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/access-control.html) を提供します。ユーザーに対して、コードレビューを完全にスキップして、集中管理リポジトリに直接pushするアクセス権を設定することが出来ます。Gerritはコードレビューなしで運用することが出来るのです。この場合、GerritはGitリポジトリとアクセス権を提供するだけです。しかし、一般的には、たとえ直接pusuするアクセス権を与えられたユーザーであっても、レビュープロセスを通してpushことがより簡単で安全な方法です。

The Life and Times of a Change

Gerritがどのように動作しているのかを理解するもっとも単純な方法は、変更内容のライフサイクルを見てみることです。一例として、稼働しているあるGerrit サーバーについて見てみましょう。サーバー名はgerrithost、プロトコルは HTTPのポートは8080、そしてSSHのポートは29418です。

現在、私達が取り組んでいるプロジェクトはRecipeBook と呼ばれています、そして私たちはマスターブランチに対する変更を開発中です。

Cloning the Repository

当たり前ですが、最初に私達がやるべきことは、修正を加えようとしているソースコードを入手することです。Gitのどのプロジェクトと同様に、Gerrit集中管理リポジトリからCloneします。

$ git clone ssh://gerrithost:29418/RecipeBook.git RecipeBook

Cloning into RecipeBook...

次に私達がやることは、私達の変更をローカルにコミットすることです。Gerritはここでは何もしません、普通に修正してgitにコミットするだけです。

必須ではありませんが、コミットする際にコミットメッセージにChange-Idを入れるのが良いでしょう。そうすると、Gerritは同じ変更(Change-Id)であればバージョンが違っていても一緒にレビューされるようにリンクすることが出来ます。Gerritには標準的な[Change-Id commit-msg hook](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/user-changeid.html) という機能が含まれているため、コミットする際にユニークなChange-Idを生成します。もしもコミットメッセージにChange-Idが含まれていない場合、Gerritは変更内容をレビューにpushする時にChenge-Idを生成します。しかしその場合、コミットメッセージにはChange-Idが含まれていないので、もしもレビューで何か指摘された場合、別のバージョンをアップロードする際には、手作業でChange-Idをコピーしなくてはいけなくなります。ですから、コミットメッセージにChange-Idを含めて、このことを忘れた方が良いのです。

Creating the Review

変更内容をローカルにコミットしたら、次はレビューしてもらうために、Gerritにプッシュします。これはGerritサーバーに対して、git pushコマンドを使います。私たちはGerritから直接ローカルにクローンしたので、Gerritがoriginです。つまり、リモートを再定義する必要はありません。

$ <work>

$ git commit

[master 9651f22] Change to a proper, yeast based pizza dough.

1 files changed, 3 insertions(+), 2 deletions(-)

$ git push origin HEAD:refs/for/master

Counting objects: 5, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 542 bytes, done.

Total 3 (delta 0), reused 0 (delta 0)

remote:

remote: New Changes:

remote: http://gerrithost:8080/68

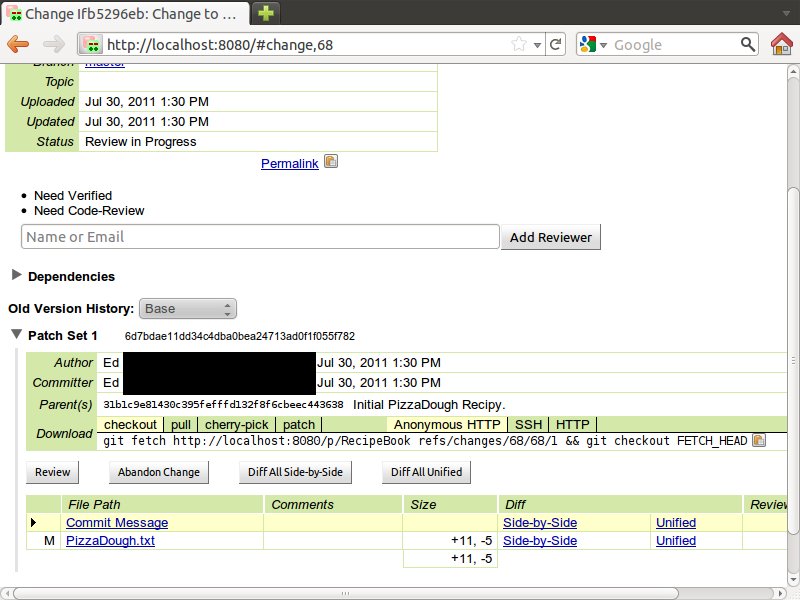
remote:

To ssh://gerrithost:29418/RecipeBook.git

\* [new branch] HEAD -> refs/for/master

通常と異なるのは、origin masterではなく、 refs/for/masterと指定している点です。これは、マジックブランチ(a magic branch)であり、実行することでmasterブランチに対するレビューを生成します。Gerrit上の全てのブランチに対して、このようのマジックブランチ refs/for/<branch\_name> を指定してプッシュすることで、レビューが生成されます。

このコマンドの実行画面をよく見ると、Gerritサーバー上のHTTPへのリンクがあることに気づきます(この場合、http://gerrithost:8080/68)。これはこのコミットに対するレビィーをするためのウェブページです。さあ、リンクを辿って、何が見えるか試してみましょう。



*Figure 3. Gerrit Code Review Screen*

これはGerrit code reviewの画面です。ここでは、誰かが変更内容をレビューしようとしています。この画面ではあまりよく見えませんが、diffを使って変更内容を比較することも出来ますし、コメントを追加して何を変更したのか、何のために変更したのか説明することも出来ます。また、レビューワーを追加することも出来ます。

レビューワーはレビューするにあたり、様々な方法で変更内容を検索することが出来ます。Gerritにはプロジェクトリーダー（もしくはそれ以外の誰に対しても）使いやすい[search](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/user-search.html)機能があります。 これを使うと、レビューしなくてはいけない変更内容を簡単に見つけることが出来ます。また、ユーザーはGerrit上のプロジェクトを検索条件([search](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/user-search.html) expression)付きでwatchすることが出来ます。設定しておけば、Gerritは検索条件にマッチした変更内容をユーザーに通知します。そのため、レビューを生成する時にレビューワーを追加すること自体は単に推奨に過ぎないのです。

この時点で、変更内容をレビューすることが可能です。この変更内容について、続きを見るために、ここで私たちの役割を変更しましょう。今から、私たちはレビューワーになります。レビューワーとして、続きを読んで下さい。

Reviewing the Change

レビューワーとしての仕事は、上に見たようなコードレビュー画面を見ることから始まります。いろいろな方法を使ってこの画面にたどり着くことが出来ますが、とにかく、私たちはこの変更内容についてレビューすると決めました。

この画面で特に注意しなくてはいけないのは、下記、２行です。

\* Need Verified

\* Need Code-Review

デフォルト設定では、Gerritは変更内容を受け入れる前にこの２つをチェックします。

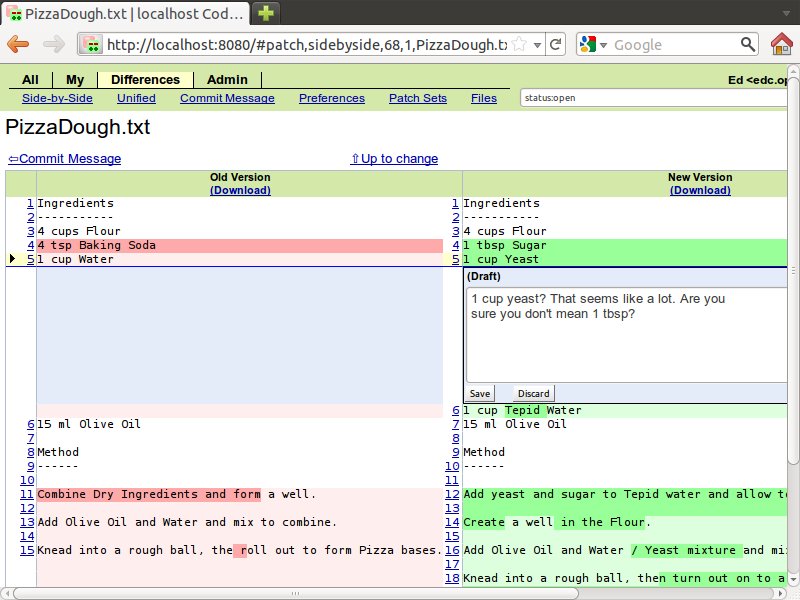
Code-Reviewは誰かがコードを見ているかどうか、プロジェクトガイドラインに準拠しているかどうか、等です。

Verifyingはコードが本当にコンパイル出来るかどうか、ユニットテストがパスするかどうか、等です。Verificationは通常は人間が実施するのではなく、自動化されたビルドサーバーによって実施されます。さらに、 [Gerrit Trigger Jenkins Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Gerrit+Trigger) を使えば、変更内容がアップロードされる度に自動的にビルドを実行してVerified scoreを更新出来ます。

GerritではCode-ReviewとVerificationについて、permissionが別々に設定されていることは注目すべき重要なことです。例えば、自動化プロセスには、Verify権限が与えらますが、code-reviewの権限を与えてはいけません。

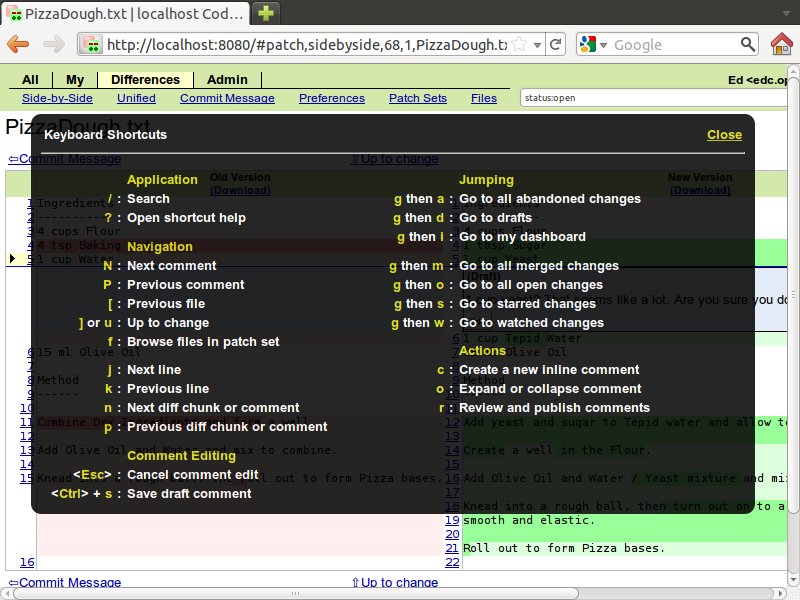
今、私たちはレビューワーなのでコードをレビューしましょう。変更内容はGerrit Webインターフェースを介して見ることが出来ます。変更内容は、ビューのオプションを選択することで、単一のdiffで見ることも出来ますし、左右比較して見比べることも出来ます。

下記の例では、左右比較ビューを選択しました。どちらのビューでも、コメントしたいコードの上でダブルクリックすることでインラインコメントを追加することが出来ます（または行番号の上で１回クリック）。また、表のヘッダー部分のどこか（”Patch Set”という言葉以外の場所であれば）をダブルクリックするか、行番号の行ヘッダー(the line-number column header)の上で１回クリックすることでファイルコメント（you can add file comment ?）を追加することも出来ます。それらのコメントを全てのユーザーに対して公開すると、変更内容について議論が開始されます。



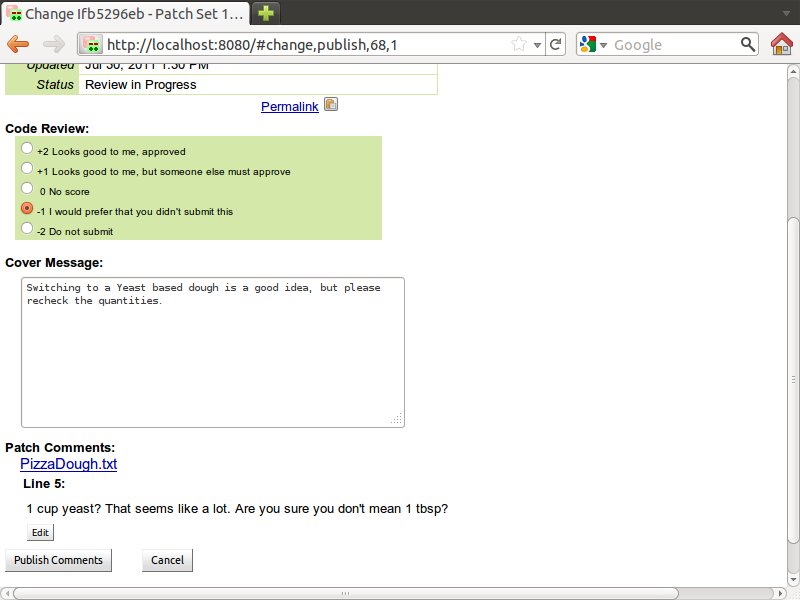
*Figure 4. Side By Side Patch View*

レビューワーは、これらの画面で多くの時間を費やすことになります。変更内容やコメントの精査を出来るだけ効率的にするために、Gerritにはほとんどのオペレーションに対応するショートカットキーが準備されています（さらには、いくつかのオペレーションはホットキーを使わないと利用出来ません）。いつでも　？（クエスチョン）キーを押せば、ショートカットキーの一覧が表示されます。



*Figure 5. Gerrit Hot Key Help*

変更内容について精査した後は、レビューを完了しなくてはいけません。そのためには、レビュー画面にある *Review*ボタンをクリックしてください。そうすると、Code Review labelとメッセージを入力することが出来ます。



*Figure 6. Reviewing the Change*

レビューワーがどのラベルを選択するかによって、次に起きることが変化します。+1と-1はただの意見に過ぎません。+2と-2は変更内容を承認するか却下するか決定します。変更内容が承認されるためには、少なくとも１つは+2がなくてはいけません、また-2が１つでもあってはいけません。これらは数字ではありますが、足し算されることはありません；+1が２つあっても+2にはなりません。

どんなラベルを選択したかに関係なく、一度*Publish Comments* ボタンがクリックされると、カバーメッセージとファイルに対するコメントが全ユーザーに対して見えるようになります。

今回、変更内容は承認されませんでした。したがって、作成者は修正する必要があります。さあ、作成者の立場に戻ってみましょう。

Reworking the Change

変更内容をアップロードする前に、[Change-Id commit-msg hook](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/user-changeid.html) を設定していたのであれば修正作業は簡単です。修正した変更内容をアップロードするためには、最初にコミットした時と同じChange-Idをコミットメッセージに付けてコミットして、プッシュすれば良いだけです。最初のコミットにはChange-Idがついているので、私たちは単にチェックアウトしてそのコミットを修正するだけです。それからGerritに対して、レビューを作成した時と同じようにプッシュすれば良いのです。例えば：

$ <checkout first commit>

$ <rework>

$ git commit --amend

$ git push origin HEAD:refs/for/master

Counting objects: 5, done.

Delta compression using up to 8 threads.

Compressing objects: 100% (2/2), done.

Writing objects: 100% (3/3), 546 bytes, done.

Total 3 (delta 0), reused 0 (delta 0)

remote: Processing changes: updated: 1, done

remote:

remote: Updated Changes:

remote: http://gerrithost:8080/68

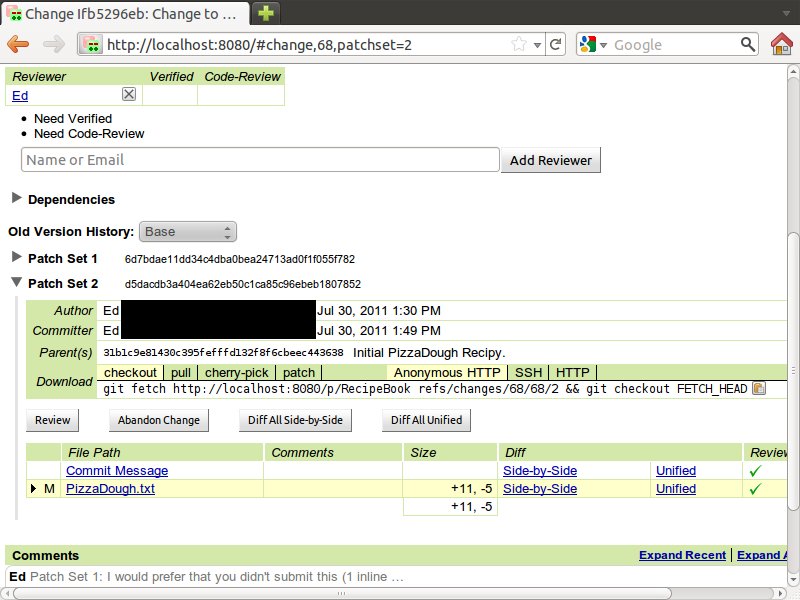
remote:

To ssh://gerrithost:29418/RecipeBook.git

\* [new branch] HEAD -> refs/for/master

今回の実行画面の結果が前回と少し違うことに気づきましたか？今回、既に存在しているレビューに対して変更内容を追加したので、変更内容が更新されたと表示されています（今回はremote: Updated Changes:、前回はremote: New Changes:）。

修正したコミットがアップロードされたので、Gerritウェブインターフェース使って、変更内容をもう一度見てみましょう。



*Figure 7. Reviewing the Rework*

よーく見てみると、この変更内容に関して、パッチが２つあるのに気づくでしょう、最初のパッチと修正されたパッチです。同じことを繰り返すのは省略して、今回は既に+2のスコアをレビューワーから受け取ったと仮定して話を進めましょう。

Trying out the Change

デフォルト設定では、Gerritは２つの署名機能があります：Code-ReviewとVerifyです。Verifyingは変更内容が本当に機能するかを確認したかどうかという意味です。これは一般的にはコードのコンパイルが通るかどうか、ユニットテストがパスするかどうか、もしくは似たようなことで確認します。実際にここでどれだけのことをチェックするのかは、プロジェクト毎に決めることが出来ます。加えて説明すると、これはGerritのデフォルト機能ですので、この機能は無効化したり、別の機能を追加したりすることも出来ます。

Code reviewセクションで述べたように、verificationは[Gerrit Trigger Jenkins Plugin](https://wiki.jenkins-ci.org/display/JENKINS/Gerrit+Trigger)もしくは似た機能を使って、一般的には自動化プロセスで実行されます。しかし、手作業でやらなくてはいけないときもあります。また、何か実際に動作しているのか、どのように動作しているのか、レビューワー自身が自分で確認しなくてはいけないときもあります。ウェブブラウザーでコードを見るよりも、開発環境でコードを実際に動かした方が良いときもあります。これら全ての場合、レビューワーは自分の開発環境に変更されたコードを拾う必要があります。Gerritではこのプロセスは簡単に出来ます。ひとつひとつの変更をgit branchとして提供します。したがって、レビューワーはGerritからそのブランチをfetch、checkoutすることで変更内容を拾うことが出来るのです。

私たちは、そんなに難しく考えることはありません。さきほどのGerrit Code Reviewの画面を見れば、そこに*download* コマンドがあったことに気づくでしょう。私たちがコードを拾うためにやらなくてはいけないことは、このコマンドをコピーペーストして、Gerritからチェックアウトするだけです。

$ git fetch http://gerrithost:8080/p/RecipeBook refs/changes/68/68/2

From http://gerrithost:8080/p/RecipeBook

\* branch refs/changes/68/68/2 -> FETCH\_HEAD

$ git checkout FETCH\_HEAD

Note: checking out 'FETCH\_HEAD'.

You are in 'detached HEAD' state. You can look around, make experimental

changes and commit them, and you can discard any commits you make in this

state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may

do so (now or later) by using -b with the checkout command again. Example:

git checkout -b new\_branch\_name

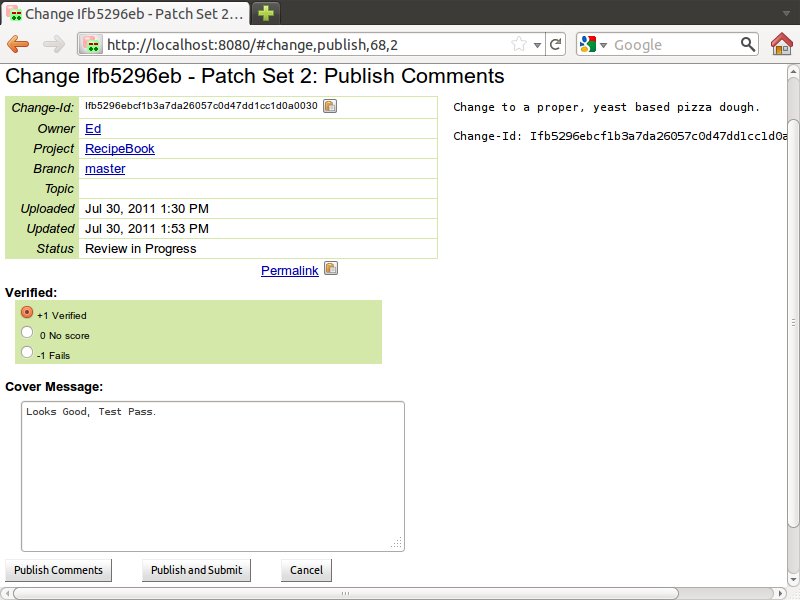
HEAD is now at d5dacdb... Change to a proper, yeast based pizza dough.

こんなに簡単に、コードが拾えました。これでレビュー中のコードをローカルに拾えたので、実行することが出来ます。あなたはもしかしたら、今まで何度も登場してきたrefspec (ここでは、refs/changes/68/68/2 )という言葉の意味について知りたいと思っているかもしれませんね。

* 最初の**68**は、変更番号(Change Idとは異なる？)を100で除算した余りです。この数字の意味は、gitリポジトリ内のディレクトリ内に置かれるファイルの数を減らすためです（.git/以下に0-99までのディレクトが生成されて、その中に変更内容の実体が分散されて配置されるので、各ディレクトリ内のファイル数は、一箇所にまとめるのに比べて1/100になる、という意味）。
* 次の **68**は、変更番号(Change Idとは異なる？)そのものです。Gir review画面をよく見ると、URLにこの数字が含まれていることに気づくでしょう。
* 最後の**2**は、その変更における、patch-setです。この例では、１番目のパッチがレビューワーから却下されたため、私たちはいくつかの修正をして２番めのパッチをアップロードしました。

Manually Verifying the Change

単純に言うと、私たちは変更内容を手作業でverifyしようと思います。Verifyする人はレビューワー自身かもしれません、もしくは全く違う人かもしれません。プロジェクトの大きさによってやり方は変わるでしょう。もしもあなたがVerify権限を持っているのであれば、Gerritのウェブインターフェースを介して、*Review*ボタンをクリックして、スコアをつけてください。



*Figure 8. Verifying the Change*

Code reviewと違い、Verifyには+2と-2のレベルは存在しません。結果はPassかfailしかありませんので、あなたが付けるべきスコアは+1:passもしくは -1:failです。

Submitting the Change

あなたは既に気づいたかもしれませんが、Verify画面のスクリーンショットには、２つのボタンがありました：*Publish Comments*と*Publish and Submit*です。

Publish and submitボタンはいつでも表示されていますが、変更内容がプロジェクトのクライテリアを既に満たしている場合だけ、クリックして正常に動作します（例えば、変更内容がverifyされて、reviewがされている場合）。そうして、ボタンを１回クリックするだけで変更内容をsubmitすると同時にレビューのスコアを付けることが出来るので便利です。もしあなたがpublish commentsをクリックした場合には、スコアは記録されますが承認はされず、集中管理リポジトリには変更内容は反映されません。この場合、*Submit Patch Set X*ボタンがメイン画面に表示されるようになります。Code-Review とVerifyが別々のオペレーションであったように、これもまた別のユーザーによってクリックされます。Submissionは３番めのオペレーションであり、実行権限はまた別のユーザーグループに制限されています。

*Publish and Submit* もしくは *Submit Patch Set X* button をクリックすると、変更内容はリポジトリにマージされます。そして、変更内容はプロジェクトの一部として正式に承認されたことになります。それ以降、誰かがgitリポジトリをfetchするとその変更が反映したコードが拾われることになります。

Part of [Gerrit Code Review](https://gerrit-documentation.storage.googleapis.com/Documentation/2.14.5.1/index.html)

 Search

Version v2.14.5.1