80

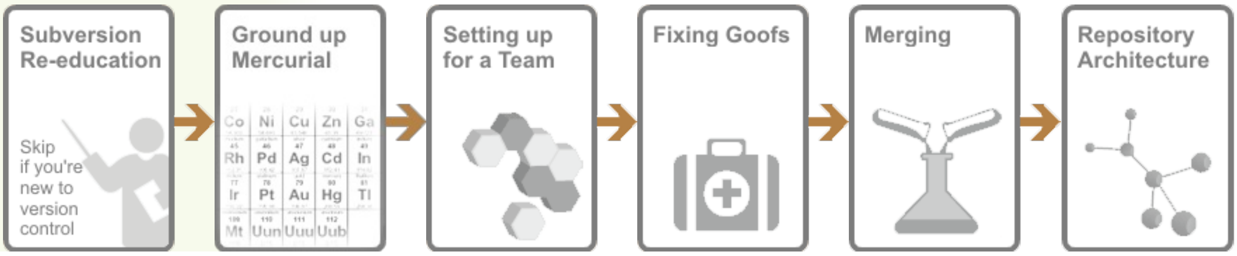
Hg

Init

200.59

Hg Init: a Mercurial tutorial

Mercurial is a modern, open source, distributed version control system, and a compelling upgrade from older systems like Subversion. In this user-friendly, six-part tutorial, [Joel Spolsky]() teaches you the key concepts. Also, Fog Creek offers [free monthly webinars]() that go over the basics of Mercurial.



If you’ve been using Subversion, Mercurial is going to be confusing. This tutorial covers the biggest differences in how Mercurial works. If you’ve never used Subversion, just skip ahead.

Subversion Re-education

When the programmers at my company decided to switch from Subversion to Mercurial, *boy* was I confused.

First, I came up with all kinds of stupid reasons why we shouldn’t switch. “We have to keep the repository on a central server, so it will be safe,” I said. You know what? I was wrong. In Mercurial every developer has a copy of the entire repository on their hard drive. It’s actually *safer*. And anyway, almost every Mercurial team uses a central repository, too, which you can back up compulsively, and you can build a three-ringed security zone complete with layers of Cylons, Stormtroopers, and adorable labradoodles (or whatever your IT department requires).

“The trouble with distributed version control is that it makes it too easy to branch,” I said, “and branching always causes problems.” Turns out this was wrong, too. I was on a streak. Branching causes problems *in Subversion* because *Subversion* doesn’t store enough information to make *merging* work. In *Mercurial,* merging is painless and easy, and so branching is commonplace and harmless.

Then I said, “Fine, I’ll use it, but don’t expect me to be able to figure it out.” And I asked Jacob to make me a cheat sheet listing all the things that I normally did in Subversion, and the equivalent way to do them in Mercurial.

Now, I could show you this cheat sheet, but I won’t, because it messed up my brain for months.

It turns out that if you’ve been using Subversion, your brain is a little bit, um, how can I say this politely? You’re brain damaged. No, that’s not polite. You need a little re-education. I walked around brain damaged for six months thinking that Mercurial was *more* complicated than Subversion, but that was only because I didn’t understand how it really worked, and once I did, it turns out—hey presto!—it’s really kind of simple.

So I wrote this tutorial for you, in which I have been very careful *not* to explain things in terms of Subversion, because there is just no reason to cause any more brain damage. The world is brain damaged enough. Instead, for those of you who are coming from Subversion, I’ve got this one chapter at the beginning that will try to reverse as much damage as possible so that you can learn Mercurial from a clean slate.

If you’ve never used Subversion, you can skip ahead to the next chapter (Ground Up Mercurial) without missing anything.

Ready? OK, let’s start with a short quiz.

Question 1. Do you write perfect code the first time?

If you answered “Yes” to question 1, you’re a liar and a cheat. You fail. Take the test again.

New code is buggy. It takes a while to get it working respectably. In the meantime, it can cause trauma for the other developers on the team.

Now, here’s how Subversion works:

- When you check new code in, everybody else gets it.

Since all new code that you write has bugs, you have a choice.

- You can check in buggy code and drive everyone else crazy, or
- You can avoid checking it in until it’s fully debugged.

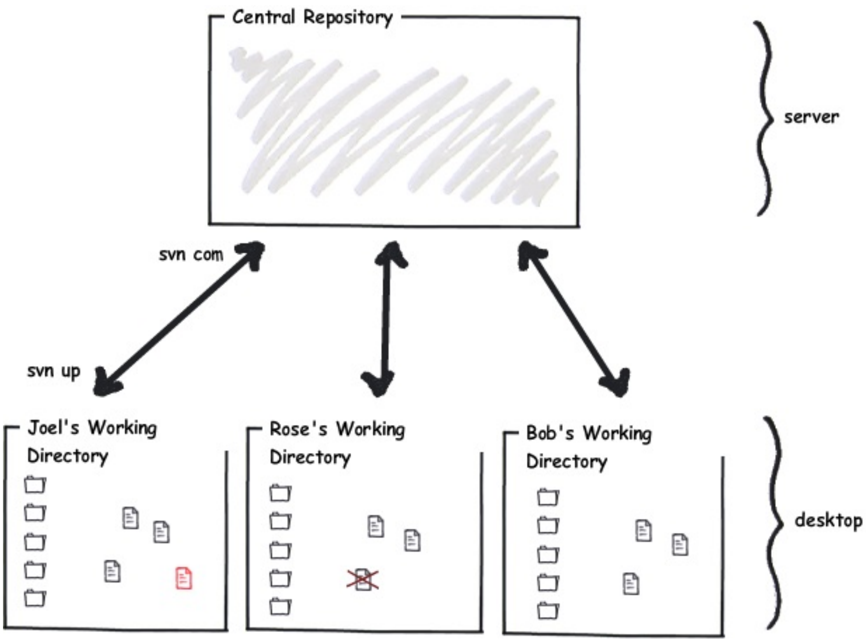
Subversion always gives you this horrible dilemma. Either the repository is full of bugs because it includes new code that was just written, *or* new code that was just written is not in the repository.

As Subversion users, we are so used to this dilemma that it’s hard to imagine it not existing.

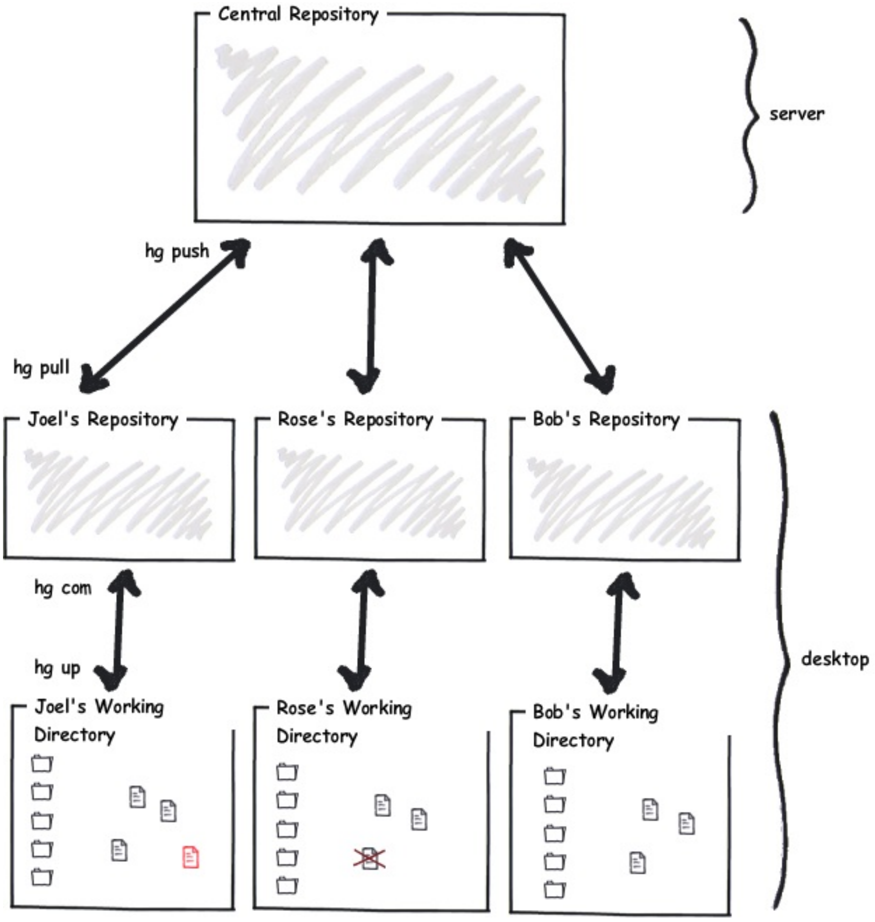
Subversion team members often go days or weeks without checking anything in. In Subversion teams, newbies are terrified of checking any code in, for fear of breaking the build, or pissing off Mike, the senior developer, or whatever. Mike once got so angry about a checkin that broke the build that he stormed into an intern’s cubicle and swept off all the contents of his desk and shouted, “This is your last day!” (It wasn’t, but the poor intern practically wet his pants.)

All this fear about checkins means people write code for weeks and weeks *without the benefit of version control* and then go find someone senior to help them check it in. Why have version control if you can’t use it?

Here’s a simple illustration of life under Subversion:



In Mercurial, *every developer has their own repository, living on their desktop:*



So you can commit your code to your private repository, and get all the benefit of version control, whenever you like. Every time you reach a logical point where your code is a little bit better, you can commit it.

Once it's solid, and you're willing to let other people use your new code, you *push* your changes from your repository to a central repository that everyone else pulls from, and they finally see your code. When it's ready.

Mercurial separates the act of committing new code from the act of inflicting it on everybody else.

And that means that you can commit (**hg com**) without anyone else getting your changes. When you've got a bunch of changes that you like that are stable and all is well, you push them (**hg push**) to the main repository.

One more big conceptual difference

You know how every single street has a name?

Well, yeah, turns out that in Japan, not so much. They usually just number the blocks in between the streets, and only very, very important streets get names.

There's a similar difference between Subversion and Mercurial.

Subversion likes to think about *revisions*. A revision is what the entire file system looked like at some particular point in time.

In Mercurial, you think about *changesets*. A changeset is a concise list of the changes between one revision and the next revision.

Six of one, half dozen of the other: what's the difference?

Here's the difference. Imagine that you and I are working on some code, and we branch that code, and we each go off into our separate workspaces and make lots and lots of changes to that code separately, so they have diverged quite a bit.

When we have to merge, Subversion tries to look at both revisions—my modified code, and your modified code—and it tries to guess how to smash them together in one big unholy mess. It usually fails, producing pages and pages of "merge conflicts" that aren't really conflicts, simply places where Subversion failed to figure out what we did.

By contrast, while we were working separately in Mercurial, Mercurial was busy keeping a *series of changesets*. And so, when we want to merge our code together, Mercurial actually has a whole lot more information: it knows *what each of us changed* and can *reapply those changes*, rather than just looking at the final product and trying to guess how to put it together.

For example, if I change a function a little bit, and then move it somewhere else, Subversion doesn't really remember those steps, so when it comes time to merge, it might think that a new function just showed up out of the blue. Whereas Mercurial will remember those things separately: function changed, function moved, which means that if you *also* changed that function a little bit, it is much more likely that Mercurial will successfully merge our changes.

Since Mercurial thinks of everything in terms of changesets, you get to do interesting things with those changesets. You can push them to a friend on the team to try them out, instead of pushing them to the central repository and inflicting them on everybody.

If this all seems a bit confusing, don't worry—by the time you get through this tutorial, it'll all make perfect sense. For now the most important thing to know is that because Mercurial thinks in terms of "changesets" instead of "revisions" *it can merge code much better than Subversion.*

And that means that *you can feel free to branch* because the merge isn't going to be a nightmare.

Want to know something funny? Almost every Subversion team I've spoken to has told me some variation on the very same story. This story is so common I should just name it "Subversion Story #1." The story is this: at some point, they tried to branch their code, usually so that the shipping version which they gave their customers can be branched off separately from the version that the developers are playing with. And every team has told me that when they tried this, it worked fine, *until they had to merge*, and then it was a nightmare. What should have been a five minute process ended up with six programmers around a single computer working for two weeks trying to manually reapply every single bug fix from the stable build back into the development build.

And almost every Subversion team told me that they vowed "never again," and they swore off branches. And now what they do is this: each new feature is in a big #ifdef block. So they can work in one single trunk, while customers never see the new code until it's debugged, and frankly, that's ridiculous.

Keeping stable and dev code separate is *precisely what source code control is* *supposed to let you do.*

When you switch to Mercurial, you may not even realize it, but branching becomes possible again, and you don't have to be afraid.

That means you can have team repositories, where a small team of programmers collaborates on a new feature, and when they're done, they merge it into the main development repository, and *it works!*

That means you can have a QA repository that lets the QA team try out the code. If it works, the QA team can push it up to the central repository, meaning, the central repository always has solid, tested code. And *it works!*

That means you can run experiments in separate repositories, and if they work, you can merge them into the main repository, and if they don't work, you can abandon them, and *it works!*

One last big conceptual difference

The last major conceptual difference between Subversion and Mercurial is not as big a deal, but it will trip you up if you're not aware of it, and here it is:

Subversion is basically revision control for *files,* but in Mercurial, revision control always applies to an entire directory—including all subdirectories.

The main way you notice this is that in Subversion, if you go into a subdirectory and commit your changes, it only commits changes in that subdirectory and all directories below it, which potentially means you've forgotten to check something in that lives in some other subdirectory which also changed. Whereas, in Mercurial, all commands always apply to the entire tree. If your code is in **c:\code**, when you issue the **hg commit** command, you can be in **c:\code** or in *any subdirectory* and it has the same effect.

This is not a big deal, but if you're used to having one big gigantic repository for the whole company, where some people only check out and work on subdirectories that they care about, this isn't a very good way to work with Mercurial—you're better off having lots of smaller repositories for each project.

And finally…

Here's the part where you're just going to have to take my word for it.

Mercurial is better than Subversion.

It is a better way of working on source code with a team. It is a better way of working on source code by yourself.

It is just *better*.

And, mark my words, if you understand the way Mercurial works, and you work the way Mercurial works, and you don't try to fight it, and you don't try to do things the old Subversion way using Mercurial but instead you learn to work the way Mercurial expects you to work, you will be happy, successful, and well-fed, and you'll always be able to find the remote control to the TV.

And in the early days, you will be tempted, I know you will, to give up on Mercurial and go back to Subversion, because it will be strange, like living in a foreign country, and you'll be homesick, and you will come up with all kinds of rationalizations, like, for example, you will claim that Mercurial working directories take up too much disk space, which is bologna, because, actually, they usually take up less space than

Subversion directories. (It’s true!)

And then you’ll go back to Subversion because you tried to branch the Subversion way, and you got confused, and it didn’t work so well, because you really should have been branching the Mercurial way, by cloning repositories, not by trying to do the things that work in Subversion, but by learning the Mercurial way of doing things, which, trust me, *works*.

And then you’ll get Jacob, or whoever the equivalent of Jacob is at your office, to give you the “Subversion to Mercurial cheat sheet,” and you’ll spend three months thinking that **hg fetch** is just like **svn up**, without really knowing what **hg fetch** does, and one day, things will go wrong and you’ll blame Mercurial, when you really should be blaming yourself for not understanding how Mercurial works.

I know you’ll do that because that is what I did.

Don’t make the same mistake. Learn Mercurial, trust Mercurial, and figure out how to do things the Mercurial way, and you will move an entire generation ahead in source code control. While your competitors are busy taking a week to resolve all the merge conflicts they got when a vendor updated a library, you’re going to type **hg merge** and say to yourself, “Oh gosh, that’s cool, it just worked.” And Mike will chill out and share a doobie with the interns, and soon it will be spring and the kids at the nearby college will trade in their heavy parkas for skimpy A&F pre-torn T-shirts, and life will be good.

« Home

Coming up: Mercurial from the ground up.

Next »

Looking for more?

Fog Creek offers free monthly webinars that go over the basics of Mercurial. Sign up at fogcreek.com.

Any questions?

Looking for help with Kiln? Check out the Kiln Knowledge Base.

About the author.

Joel Spolsky is the founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they’re delighted. Fog Creek makes FogBugz, Kiln, and Fog Creek Copilot. Joel's blog Joel on Software is read by programmers everywhere.

80
Hg
Init
200.59

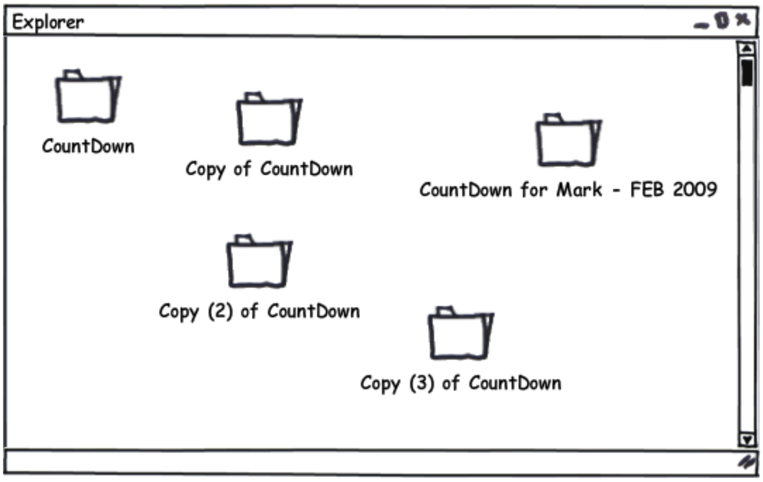
Even if you’re working by yourself, you should use Mercurial to get the benefits of version control. This tutorial shows how easy it is to check a directory into Mercurial so you can track old versions easily.

Ground up Mercurial

Mercurial is a *version control system*. Developers use it to manage source code. It serves two important purposes:

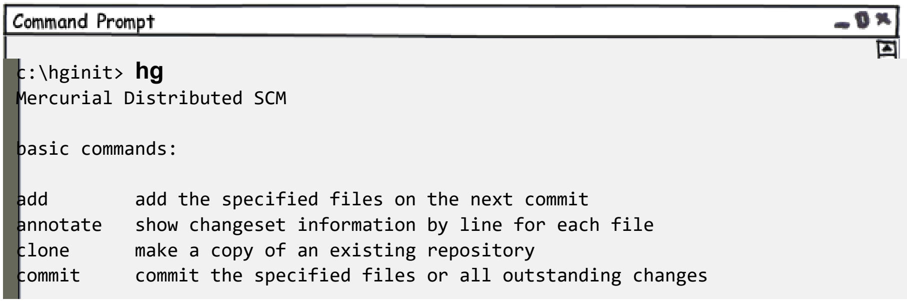
1. It keeps track of every old version of every file
2. It can merge different versions of your code, so that teammates can work independently on the code and then merge their changes

Without Mercurial, you could try to keep old versions just by making a lot of copies of the directory containing all your code:

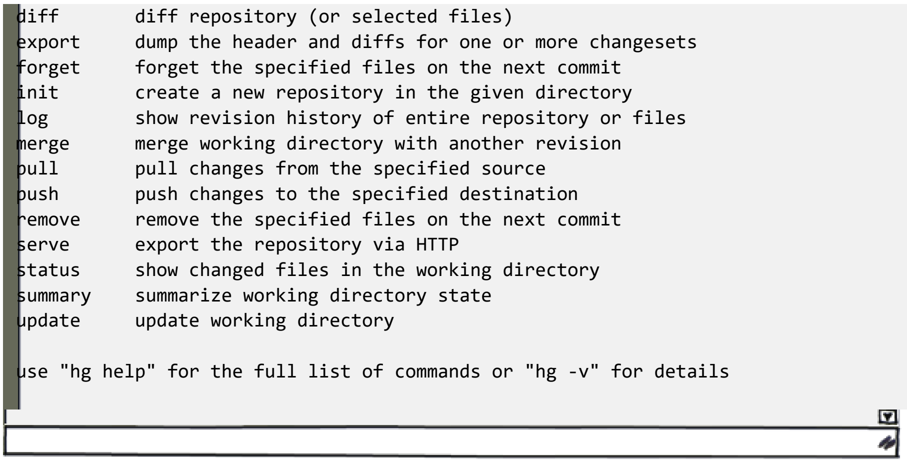


This is tedious, takes up a lot of disk space, and confusing. Using version control is a better way to do this.

Most people work with Mercurial through the command line, which works on Windows, Unix, and Mac. The command for Mercurial is **hg**:



```
c:\hginit> hg
Mercurial Distributed SCM

basic commands:

add        add the specified files on the next commit
annotate   show changeset information by line for each file
clone      make a copy of an existing repository
commit     commit the specified files or all outstanding changes
```

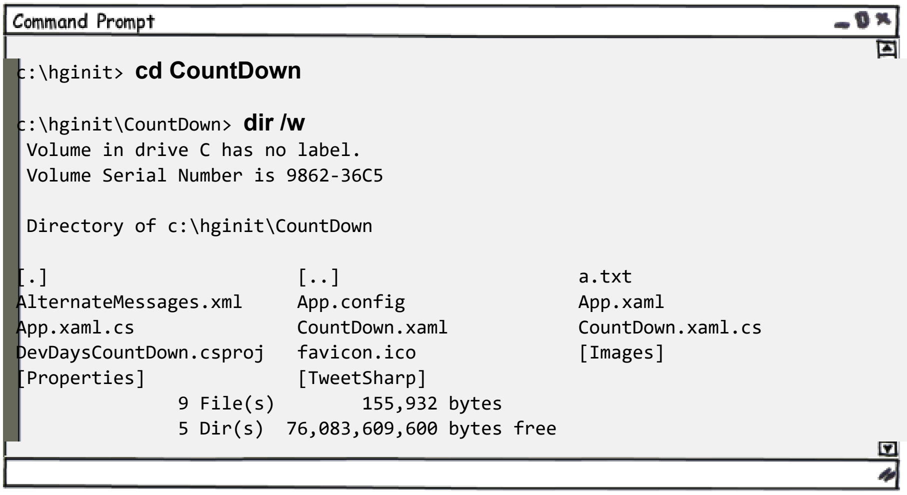


```
diff       diff repository (or selected files)
export     dump the header and diffs for one or more changesets
forget     forget the specified files on the next commit
init       create a new repository in the given directory
log        show revision history of entire repository or files
merge      merge working directory with another revision
pull       pull changes from the specified source
push       push changes to the specified destination
remove     remove the specified files on the next commit
serve      export the repository via HTTP
status     show changed files in the working directory
summary    summarize working directory state
update     update working directory

use "hg help" for the full list of commands or "hg -v" for details
```

Typing **hg** without anything else gives you a list of the most common commands that are available. You can also try **hg help** for a complete list of commands.

To take advantage of version control, you needed a *repository*. A repository stores all your old versions of every file. To save disk space, it's not actually going to store every old version—it's just going to store a compact list of changes.

In the old days, getting a repository was a big deal. You had to have a central server somewhere and you had to install software on it. Mercurial is *distributed*, so you can use it without a fancy central server. You can run it entirely on your own computer. And getting a repository is super-easy: you just go into the top-level directory where all your code lives...

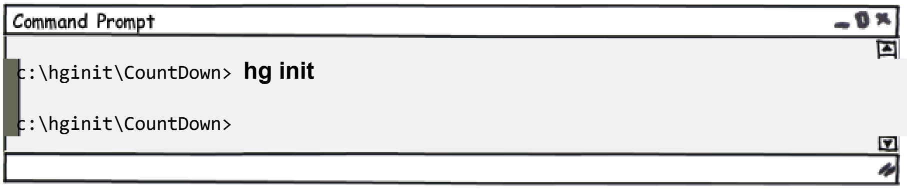


```
c:\hginit> cd CountDown

c:\hginit\CountDown> dir /w
 Volume in drive C has no label.
 Volume Serial Number is 9862-36C5

 Directory of c:\hginit\CountDown

[.]                       [..]                      a.txt
AlternateMessages.xml     App.config                App.xaml
App.xaml.cs               CountDown.xaml            CountDown.xaml.cs
DevDaysCountDown.csproj   favicon.ico               [Images]
[Properties]              [TweetSharp]
               9 File(s)        155,932 bytes
               5 Dir(s)  76,083,609,600 bytes free
```

... there's my code, and you type **hg init**:

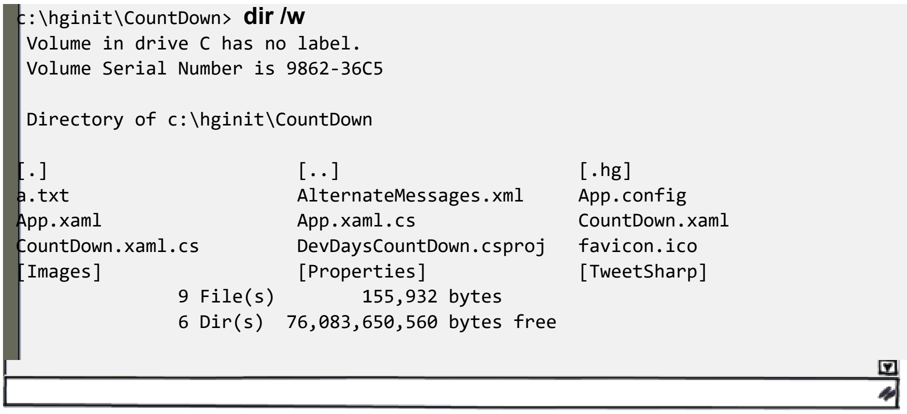
hg init

creates a repository



```
c:\hginit\CountDown> hg init

c:\hginit\CountDown>
```

Wait a minute, did anything happen? It doesn't look like anything happened. But if you look closely, you'll see that there's a new directory there, named **.hg**:



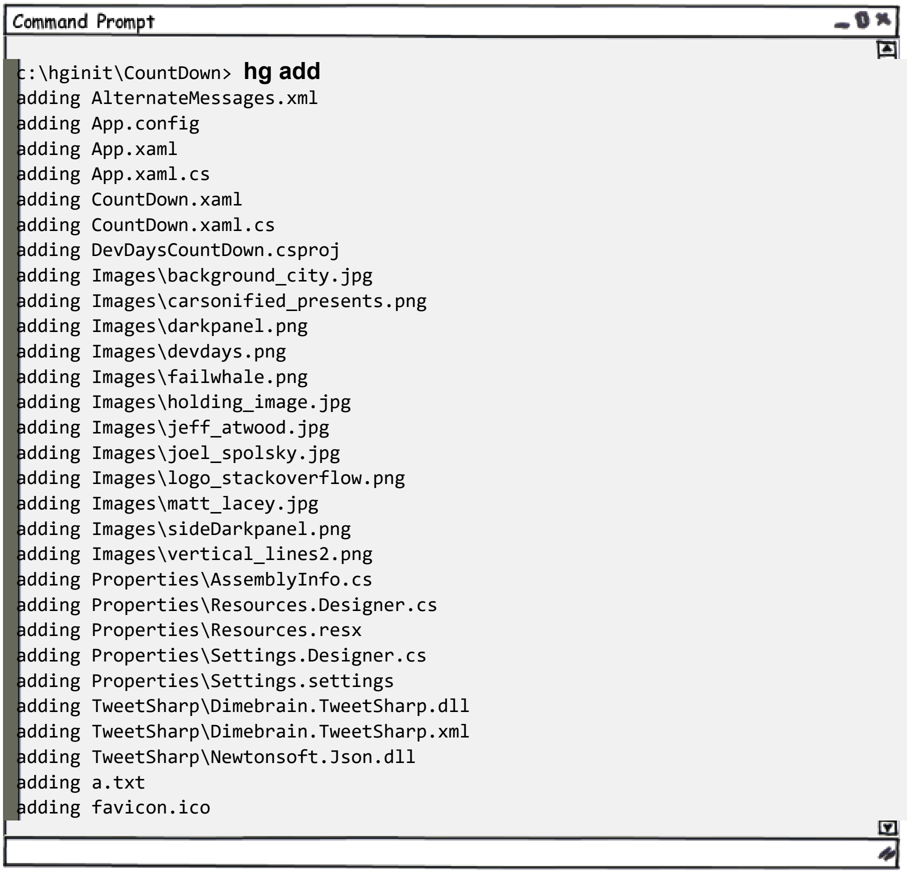
```
c:\hginit\CountDown> dir /w
 Volume in drive C has no label.
 Volume Serial Number is 9862-36C5

 Directory of c:\hginit\CountDown

[.]                        [..]                       [.hg]
a.txt                      AlternateMessages.xml      App.config
App.xaml                   App.xaml.cs                CountDown.xaml
CountDown.xaml.cs          DevDaysCountDown.csproj    favicon.ico
[Images]                   [Properties]               [TweetSharp]
               9 File(s)        155,932 bytes
               6 Dir(s)  76,083,650,560 bytes free
```

That's the repository! It's a directory full of everything Mercurial needs. Settings, old version of files, tags, an extra pair of socks for when it rains, etc. *Don't go in there.* You are almost never going to want to mess with that directory directly.

OK, now that we have a fresh new repository, we're going to want to add all these source files to it. That's easy, too: just type **hg add**.

hg add

schedules files to be added to the repository. They won't actually be added until you commit



```
c:\hginit\CountDown> hg add
adding AlternateMessages.xml
adding App.config
adding App.xaml
adding App.xaml.cs
adding CountDown.xaml
adding CountDown.xaml.cs
adding DevDaysCountDown.csproj
adding Images\background_city.jpg
adding Images\carsonified_presents.png
adding Images\darkpanel.png
adding Images\devdays.png
adding Images\failwhale.png
adding Images\holding_image.jpg
adding Images\jeff_atwood.jpg
adding Images\joel_spolsky.jpg
adding Images\logo_stackoverflow.png
adding Images\matt_lacey.jpg
adding Images\sideDarkpanel.png
adding Images\vertical_lines2.png
adding Properties\AssemblyInfo.cs
adding Properties\Resources.Designer.cs
adding Properties\Resources.resx
adding Properties\Settings.Designer.cs
adding Properties\Settings.settings
adding TweetSharp\Dimebrain.TweetSharp.dll
adding TweetSharp\Dimebrain.TweetSharp.xml
adding TweetSharp\Newtonsoft.Json.dll
adding a.txt
adding favicon.ico
```

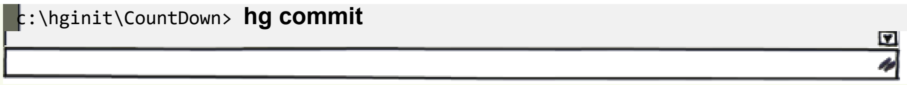
There's still one more step… you have to *commit* your changes. What changes? The change of adding all those files.

Why do you have to commit? With Mercurial, committing says "hey, the way the files look right now—please remember that." It's like making a copy of the whole directory… every time you have something that you've changed that you sorta like, you commit.

hg commit

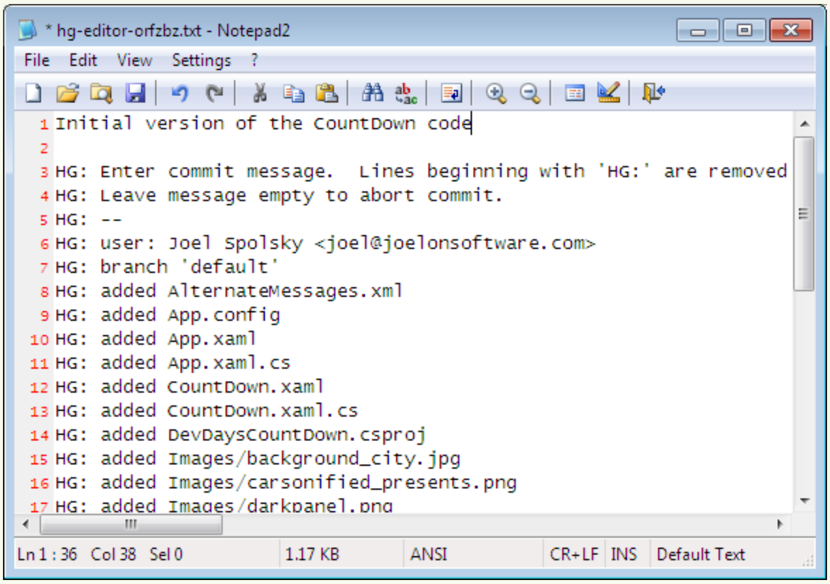
saves the current state of all files to the

repository



```
c:\hginit\CountDown> hg commit
```

Mercurial will pop up an editor so that you can type a commit message. This is just something you type to remind yourself of what changed in this commit.



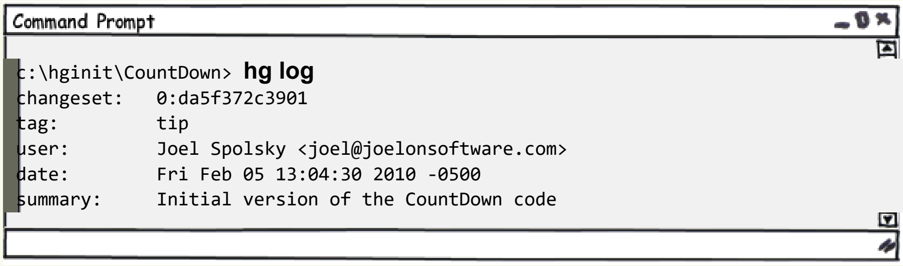
```
Initial version of the CountDown code

HG: Enter commit message.  Lines beginning with 'HG:' are removed
HG: Leave message empty to abort commit.
HG: --
HG: user: Joel Spolsky <joel@joelonsoftware.com>
HG: branch 'default'
HG: added AlternateMessages.xml
HG: added App.config
HG: added App.xaml
HG: added App.xaml.cs
HG: added CountDown.xaml
HG: added CountDown.xaml.cs
HG: added DevDaysCountDown.csproj
HG: added Images/background_city.jpg
HG: added Images/carsonified_presents.png
HG: added Images/darkpanel.png
```

When you save and exit, your files will be committed.

hg log

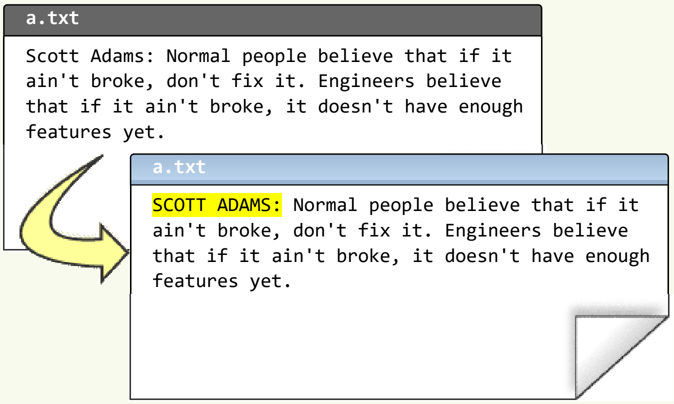
shows the history of changes committed to the repository

You can type **hg log** to see a history of changes. It's like your repository's blog:



```
c:\hginit\CountDown> hg log
changeset:   0:da5f372c3901
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:04:30 2010 -0500
summary:     Initial version of the CountDown code
```

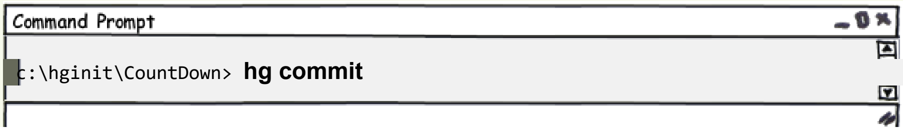
Let's edit a file and see what happens.



```
Scott Adams: Normal people believe that if it
ain't broke, don't fix it. Engineers believe
that if it ain't broke, it doesn't have enough
features yet.
```

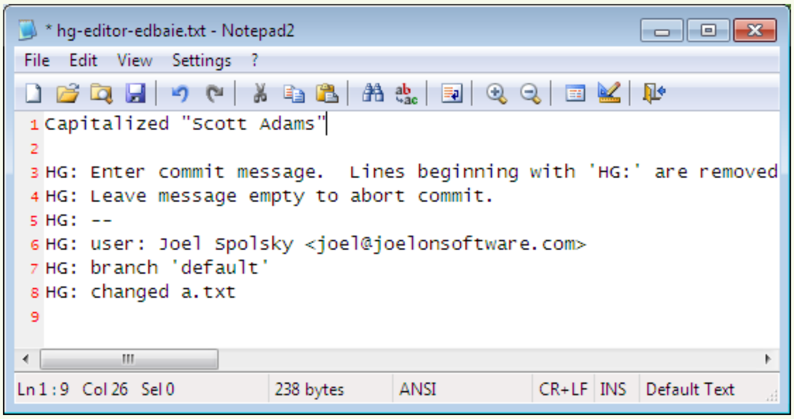
```
SCOTT ADAMS: Normal people believe that if it
ain't broke, don't fix it. Engineers believe
that if it ain't broke, it doesn't have enough
features yet.
```

Now that we've made another change, we can commit it using **hg commit**:



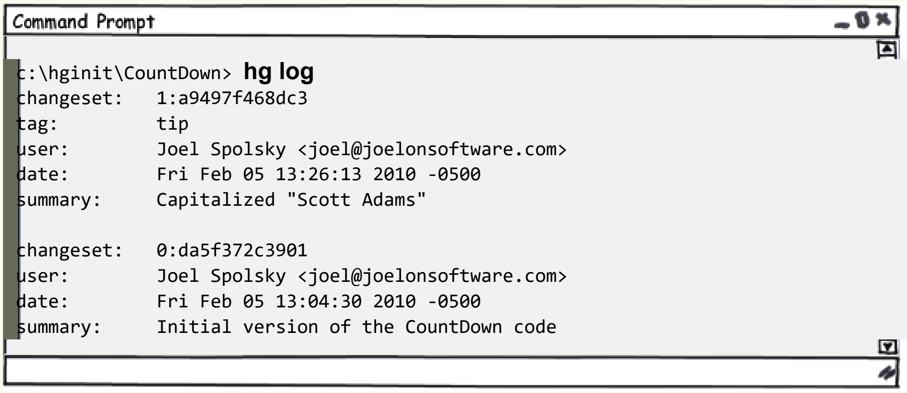
```
c:\hginit\CountDown> hg commit
```

Notice that Mercurial has figured out that only one file, a.txt, changed:



```
Capitalized "Scott Adams"

HG: Enter commit message.  Lines beginning with 'HG:' are removed
HG: Leave message empty to abort commit.
HG: --
HG: user: Joel Spolsky <joel@joelonsoftware.com>
HG: branch 'default'
HG: changed a.txt

```

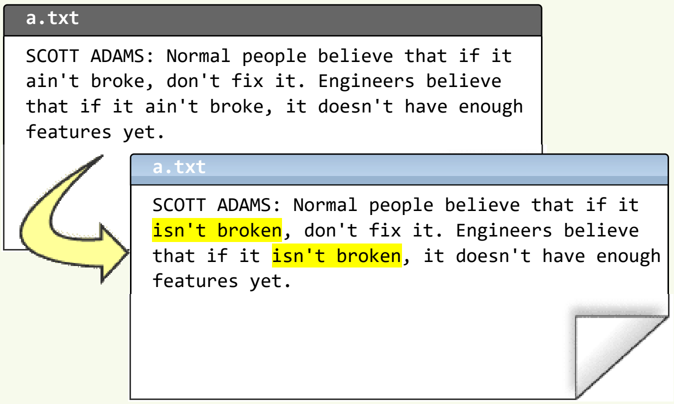
And now that I've committed, let's take a look at the log:



```
c:\hginit\CountDown> hg log
changeset:   1:a9497f468dc3
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:26:13 2010 -0500
summary:     Capitalized "Scott Adams"

changeset:   0:da5f372c3901
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:04:30 2010 -0500
summary:     Initial version of the CountDown code
```

Like any modern blogger, Mercurial puts the newest stuff on top.

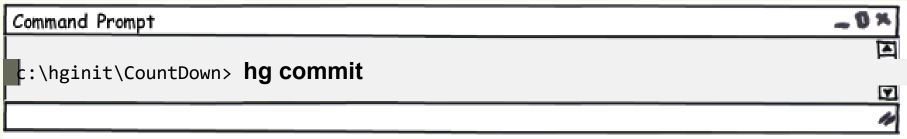
I'm going to make one more change, just to amuse myself.



```
SCOTT ADAMS: Normal people believe that if it
ain't broke, don't fix it. Engineers believe
that if it ain't broke, it doesn't have enough
features yet.
```

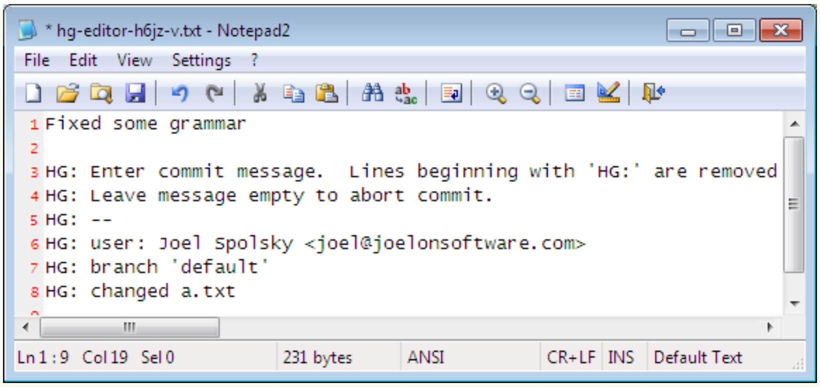
```
SCOTT ADAMS: Normal people believe that if it
isn't broken, don't fix it. Engineers believe
that if it isn't broken, it doesn't have enough
features yet.
```

Committing:



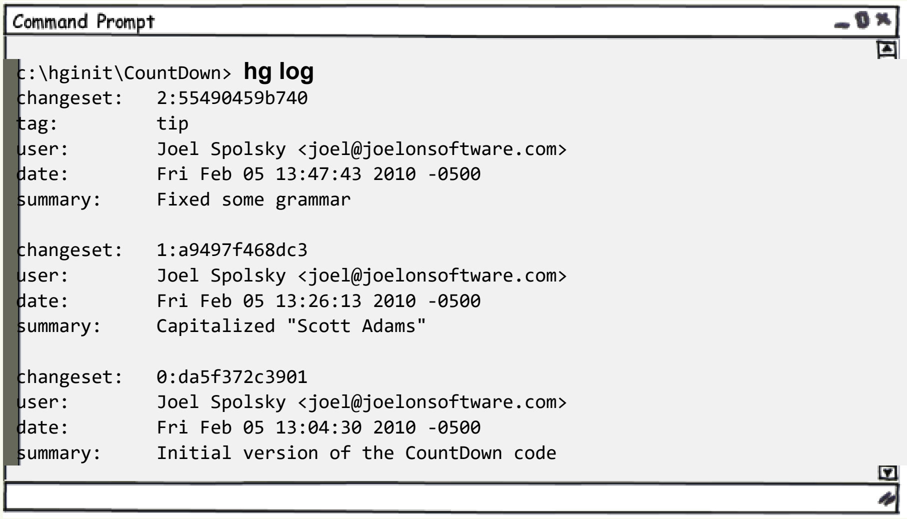
```
c:\hginit\CountDown> hg commit
```

My commit message:



```
Fixed some grammar

HG: Enter commit message.  Lines beginning with 'HG:' are removed
HG: Leave message empty to abort commit.
HG: --
HG: user: Joel Spolsky <joel@joelonsoftware.com>
HG: branch 'default'
HG: changed a.txt
```

And now what does the log show?



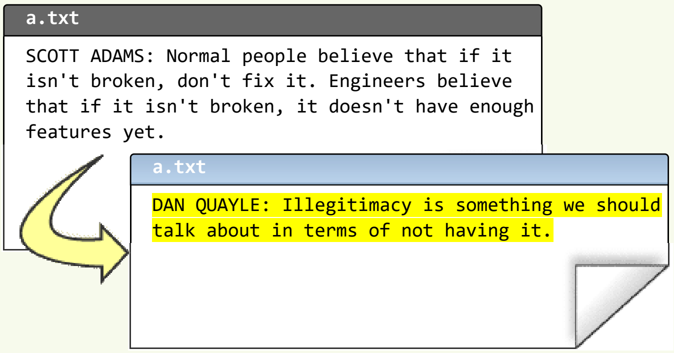
```
c:\hginit\CountDown> hg log
changeset:   2:55490459b740
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:47:43 2010 -0500
summary:     Fixed some grammar

changeset:   1:a9497f468dc3
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:26:13 2010 -0500
summary:     Capitalized "Scott Adams"

changeset:   0:da5f372c3901
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:04:30 2010 -0500
summary:     Initial version of the CountDown code
```

OK, that was a lot of fun. I made some changes, and each time I made a significant change, I committed it to the repository.

I know what you're thinking. You're thinking, "JOEL, THIS ALL SEEMS LIKE A BIG WASTE OF TIME." Why go through all this rigamarole of committing?

Patience, young grasshopper. You're about to learn how to get some benefit out of this.

Number one. Let's say you make a huge mistake editing.



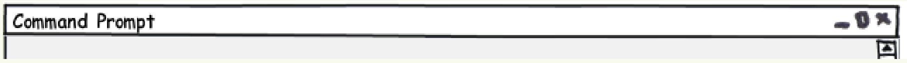
a.txt

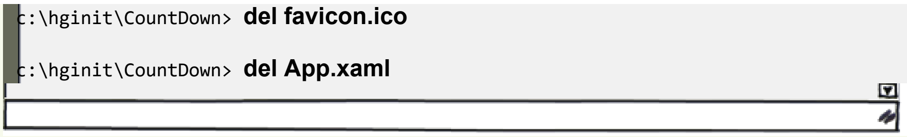
SCOTT ADAMS: Normal people believe that if it isn't broken, don't fix it. Engineers believe that if it isn't broken, it doesn't have enough features yet.

a.txt

DAN QUAYLE: Illegitimacy is something we should talk about in terms of not having it.

And then, gosh, just for good measure you delete a couple of really important files.





```
c:\hginit\CountDown> del favicon.ico

c:\hginit\CountDown> del App.xaml
```

In the days before Mercurial, this would be a good opportunity to go crying to the system administrator and asking piercingly sad questions about *why* the backup system is “temporarily” out of commission and has been for the last eight months.

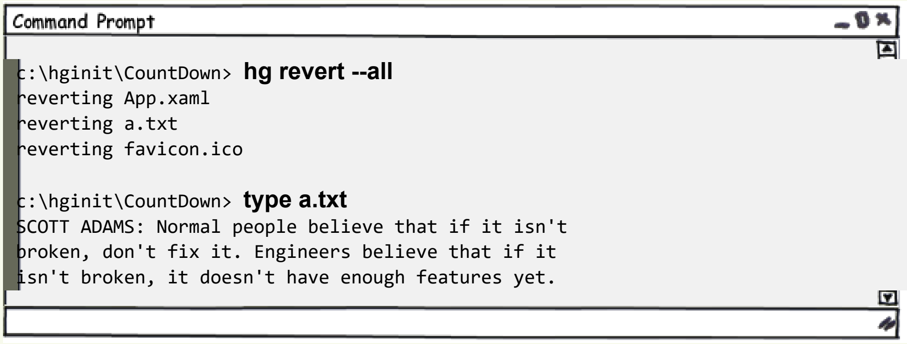
The system administrator, whom everybody calls Taco, is too shy to eat lunch with the rest of the team. On those rare occasions where he is away from his swivel office chair, you will notice a triangular-shaped salsa-colored stain on the seat where drippings of his many Mexican lunches fell between his legs, insuring that nobody takes his chair, even though it is the superior Herman Miller variety that the company founders bought themselves, not the standard-issue Staples $99 special that causes everyone else back pain.

Anyway, yeah, there’s no backup.

Thanks to Mercurial, though, when you’re unhappy with your changes, you can just issue the handy command **hg revert** which will immediately revert your directory back to the way it was at the time of the last commit.

hg revert

revert changed files back to committed version



```
c:\hginit\CountDown> hg revert --all
reverting App.xaml
reverting a.txt
reverting favicon.ico

c:\hginit\CountDown> type a.txt
SCOTT ADAMS: Normal people believe that if it isn't
broken, don't fix it. Engineers believe that if it
isn't broken, it doesn't have enough features yet.
```

I used the command line argument **--all** because I wanted to revert *all* files back to their original state.

So, when you’re working on source code with Mercurial:

1. Make some changes
2. See if they work
3. If they do, **commit** them
4. If they don’t, **revert** them
5. GOTO 1

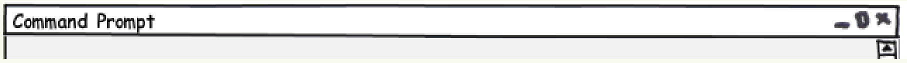
(I know. Between the Windows command prompt and the GOTO statements, I am the *least cool programmer* who ever lived.)

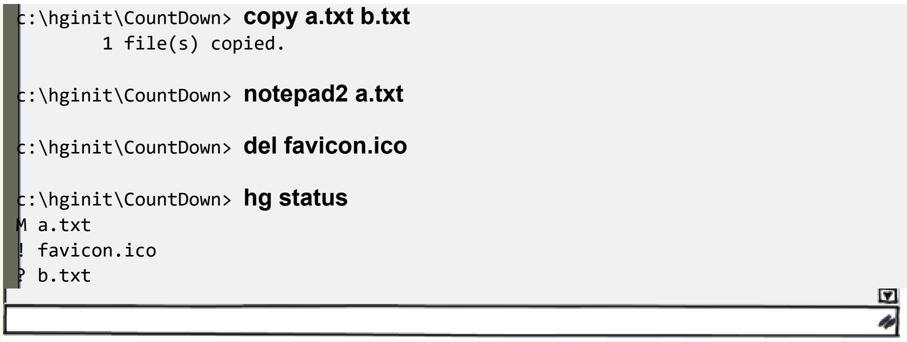
hg status

shows a list of changed files

As time goes on, you may get confused about where you are and what changes you’ve made since the last commit. Mercurial keeps track of all that for you. All you have to do is type **hg status** and Mercurial will give you a list of files that have changed.

Suppose I create a file, edit a file, and delete a file.





```
c:\hginit\CountDown> copy a.txt b.txt
        1 file(s) copied.

c:\hginit\CountDown> notepad2 a.txt

c:\hginit\CountDown> del favicon.ico

c:\hginit\CountDown> hg status
M a.txt
! favicon.ico
? b.txt
```

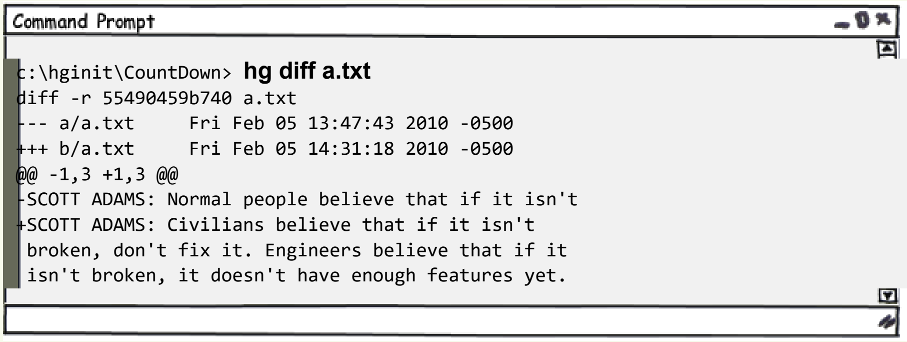
hg status lists any files that have changed with a little letter at the beginning of the line telling you what's up. "M" means "Modified"—the file has been changed. "!" means missing—the file is supposed to be there, but it disappeared. "?" means unknown—Mercurial doesn't know anything about this file. Yet.

Let's deal with these changes one at a time. That modified file, **a.txt**. What's modified about it? You may have forgotten what you changed! Heck, I can barely even remember what I ate for breakfast most days. Which is especially worrisome because it's ALWAYS CHEERIOS. Anyway, a.txt has changed. What changed?

There's a command for that: **hg diff** tells you exactly what's changed with a file since the last commit.

hg diff

shows what changed in a file



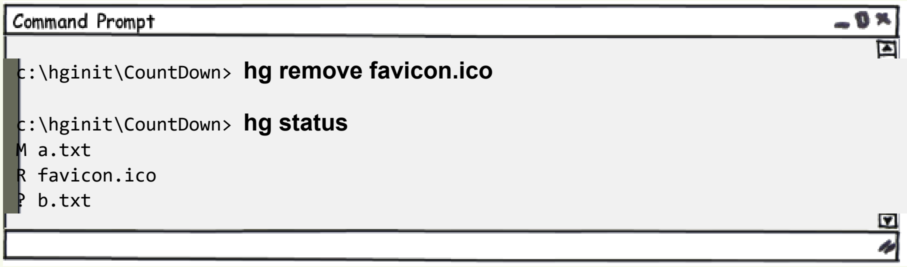
```
c:\hginit\CountDown> hg diff a.txt
diff -r 55490459b740 a.txt
--- a/a.txt     Fri Feb 05 13:47:43 2010 -0500
+++ b/a.txt     Fri Feb 05 14:31:18 2010 -0500
@@ -1,3 +1,3 @@
-SCOTT ADAMS: Normal people believe that if it isn't
+SCOTT ADAMS: Civilians believe that if it isn't
 broken, don't fix it. Engineers believe that if it
 isn't broken, it doesn't have enough features yet.
```

This format is a little bit cryptic, but the interesting part is that you can see some lines that begin with a minus, which were removed, and lines that begin with a plus, which were added, so you can see here that "Normal people" was edited to be "Civilians".

hg remove

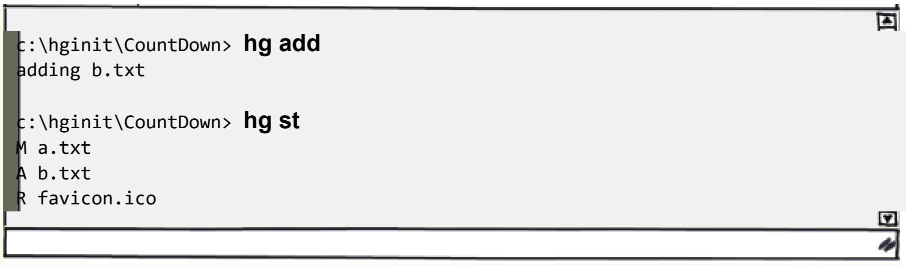
schedules files to be removed from the repository. They won't actually be removed until you commit.

Now. That missing file, favicon.ico. As earlier, if you didn't mean to delete it, you can **hg revert**, but let's assume you really did mean to remove it. Whenever you remove (or add) a file, you have to tell Mercurial:



```
c:\hginit\CountDown> hg remove favicon.ico

c:\hginit\CountDown> hg status
M a.txt
R favicon.ico
? b.txt
```

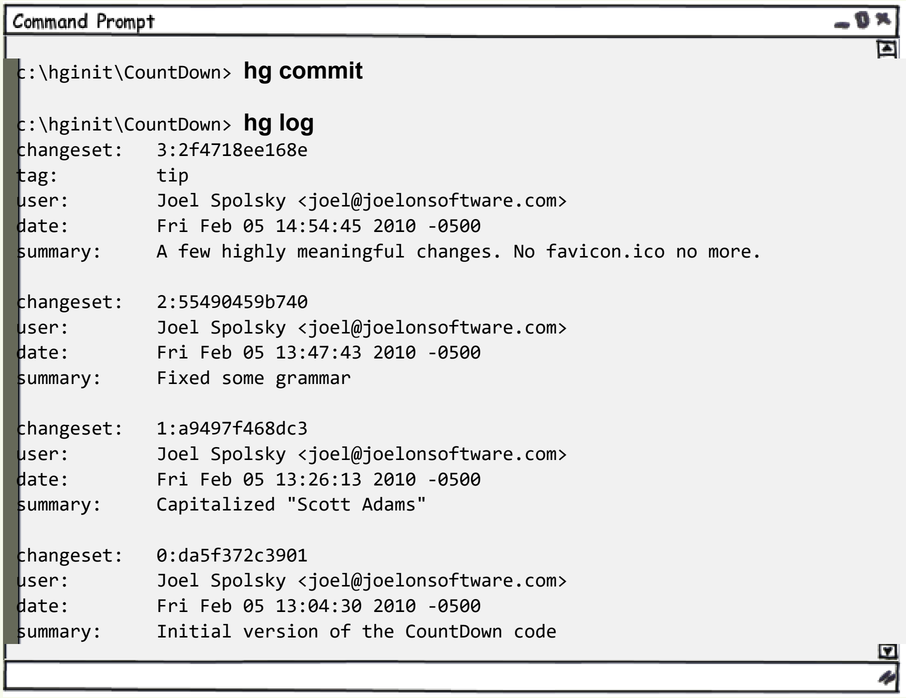
The "R" means "Removed" so the next time we commit in Mercurial this file will be removed. (The *history* of the file will remain in the repository, so of course we can always get it back). Finally, we need to add that new file, **b.txt**:



```
c:\hginit\CountDown> hg add
adding b.txt

c:\hginit\CountDown> hg st
M a.txt
A b.txt
R favicon.ico
```

“A” means “Added.” Did you notice I was getting lazy about typing out **hg status** all the time? Mercurial only needs enough letters to disambiguate, and there are no other commands that start with **st**.

Having solved all the little ?’s and !’s, I can go ahead and check in my changes:



```
c:\hginit\CountDown> hg commit

c:\hginit\CountDown> hg log
changeset:   3:2f4718ee168e
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 14:54:45 2010 -0500
summary:     A few highly meaningful changes. No favicon.ico no more.

changeset:   2:55490459b740
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:47:43 2010 -0500
summary:     Fixed some grammar

changeset:   1:a9497f468dc3
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:26:13 2010 -0500
summary:     Capitalized "Scott Adams"

changeset:   0:da5f372c3901
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 05 13:04:30 2010 -0500
summary:     Initial version of the CountDown code
```

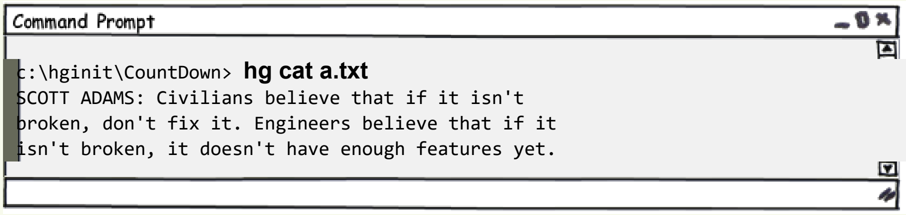
One more thing about the output from **hg log**: the *changeset* line shows us a number to every commit.... actually two numbers: a handy, short one, like “0” for your initial revision, etc., and a long, goofy hexadecimal one which you can ignore for now.

Remember that Mercurial keeps, in the repository, enough information to reconstruct any old version of any file.

hg cat

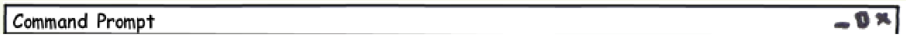
shows any revision of any file.

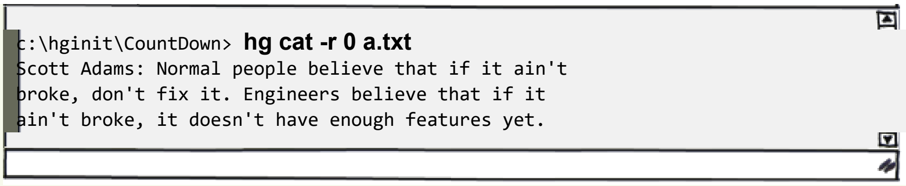
First of all, the simple command **hg cat** can be used to print any old version of a file. For example, to see what a.txt looks like now:



```
c:\hginit\CountDown> hg cat a.txt
SCOTT ADAMS: Civilians believe that if it isn't
broken, don't fix it. Engineers believe that if it
isn't broken, it doesn't have enough features yet.
```

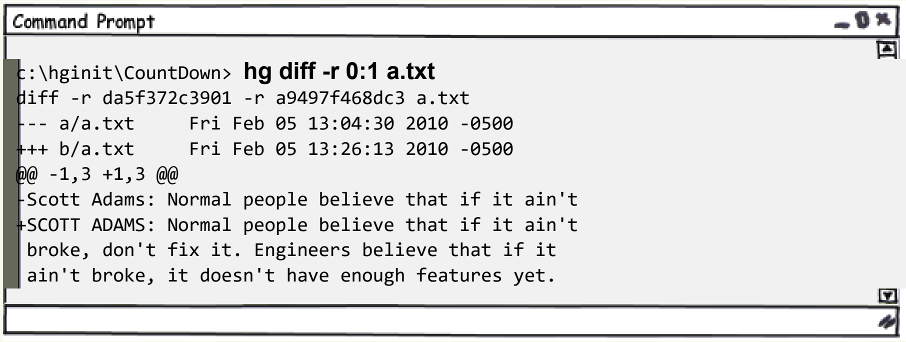
To see what it looked like in the past, I can just pick a changeset number from the log. Then I use the **cat** command with the **-r** (“revision”) argument:





```
c:\hginit\CountDown> hg cat -r 0 a.txt
Scott Adams: Normal people believe that if it ain't
broke, don't fix it. Engineers believe that if it
ain't broke, it doesn't have enough features yet.
```

If the file is long and complicated, and only a little bit of it changed, I can use the **hg diff** command with an **-r** argument to print the difference between any two revisions. For example, to see what changed between revisions 0 and 1:



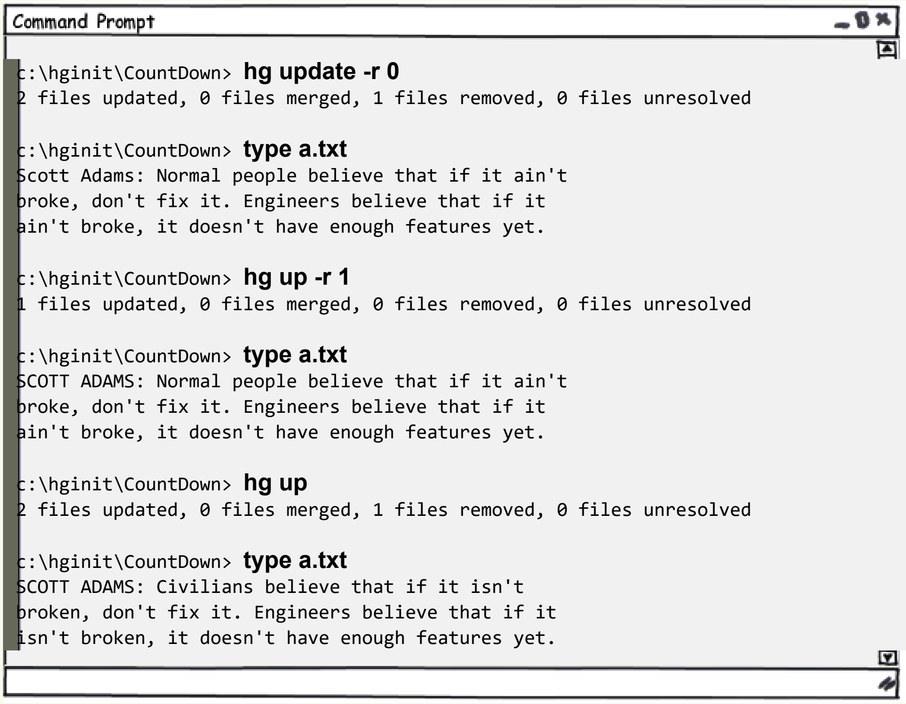
```
c:\hginit\CountDown> hg diff -r 0:1 a.txt
diff -r da5f372c3901 -r a9497f468dc3 a.txt
--- a/a.txt     Fri Feb 05 13:04:30 2010 -0500
+++ b/a.txt     Fri Feb 05 13:26:13 2010 -0500
@@ -1,3 +1,3 @@
-Scott Adams: Normal people believe that if it ain't
+SCOTT ADAMS: Normal people believe that if it ain't
 broke, don't fix it. Engineers believe that if it
 ain't broke, it doesn't have enough features yet.
```

Finally, if you haven't collapsed yet from exhaustion, before I finish this tutorial, I just want to show you *one more tiny thing:* you can use the **hg update** command to go backwards or forwards in time to any revision you want. Well, you can't really go into the future *per se*, although that would be super-cool. If you only had four revisions you would just **hg update -r 103994** and get some really cool anti-gravity sci-fi futuristic version of your source code. But of course, that is not possible.

hg update

update the working directory to a particular revision

What is possible is going *back* to any version. Watch:



```
c:\hginit\CountDown> hg update -r 0
2 files updated, 0 files merged, 1 files removed, 0 files unresolved

c:\hginit\CountDown> type a.txt
Scott Adams: Normal people believe that if it ain't
broke, don't fix it. Engineers believe that if it
ain't broke, it doesn't have enough features yet.

c:\hginit\CountDown> hg up -r 1
1 files updated, 0 files merged, 0 files removed, 0 files unresolved

c:\hginit\CountDown> type a.txt
SCOTT ADAMS: Normal people believe that if it ain't
broke, don't fix it. Engineers believe that if it
ain't broke, it doesn't have enough features yet.

c:\hginit\CountDown> hg up
2 files updated, 0 files merged, 1 files removed, 0 files unresolved

c:\hginit\CountDown> type a.txt
SCOTT ADAMS: Civilians believe that if it isn't
broken, don't fix it. Engineers believe that if it
isn't broken, it doesn't have enough features yet.
```

hg update is actually modifying every file in the directory that changed to go backwards and forwards through time. If a file was added or removed, it adds or removes it. Without any arguments, **hg update** goes to the latest version.

Test yourself

OK! That’s it for tutorial one. Here’s all the things you should know how to do now:

1. Create a repository
2. Add and remove files in a repository
3. After making changes, see what uncommitted changes you made, then
4. … commit if you like them,
5. … or revert if you don’t.
6. See old versions of files, or even move your directory backwards and forwards in time

« Home

Next, we’ll see how to get your whole team working with Mercurial.

Next »

Any questions?

If you have any questions about the material in this tutorial, no matter how newbie, ask them at the Kiln Knowledge Exchange.

About the author.

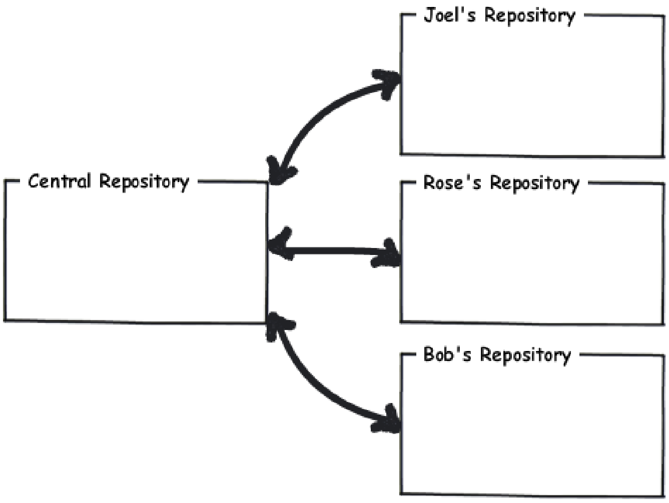
Joel Spolsky is the founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they’re delighted. Fog Creek makes FogBugz, Kiln, and Fog Creek Copilot. Joel's blog Joel on Software is read by programmers everywhere.

80
Hg
Init
200.59

One of the benefits of using Mercurial is working with a team on the same code. Mercurial lets you each work independently, and merges your changes together.

Setting up for a Team

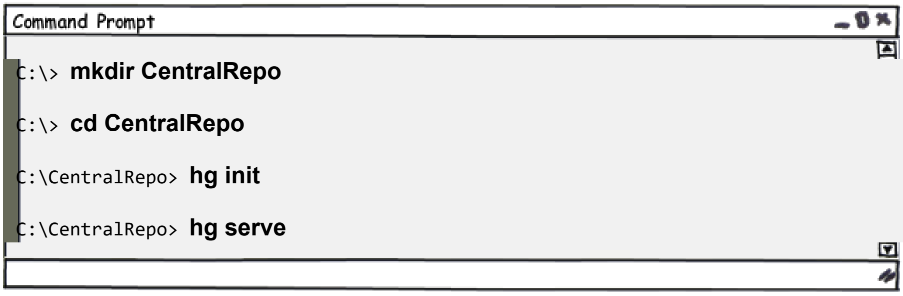
The most common way to collaborate with Mercurial is to set up a central repository, in addition to the private repositories that we each have on our own computers. We can use the central repository sort of like a swap meet, where we get together to trade the changes we've been making.



hg serve

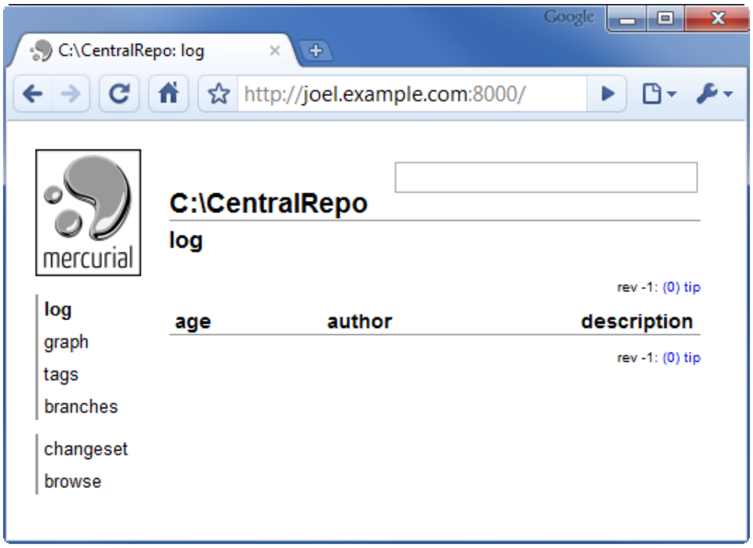
runs a web server to make the current repository accessible over the Internet

The quick-and-dirty way to make a central repository is to use Mercurial's built in web server—all you have to do is make a repository with **hg init** and then serve it on the web with **hg serve**. By default it will be served on port 8000.



```
C:\> mkdir CentralRepo

C:\> cd CentralRepo

C:\CentralRepo> hg init

C:\CentralRepo> hg serve
```

Since this computer is named **joel.example.com** I can just go to **http://joel.example.com:8000/** with a web browser and see that the server is up and running, even though the repository there is completely empty.



C:\CentralRepo

log

rev -1: (0) tip

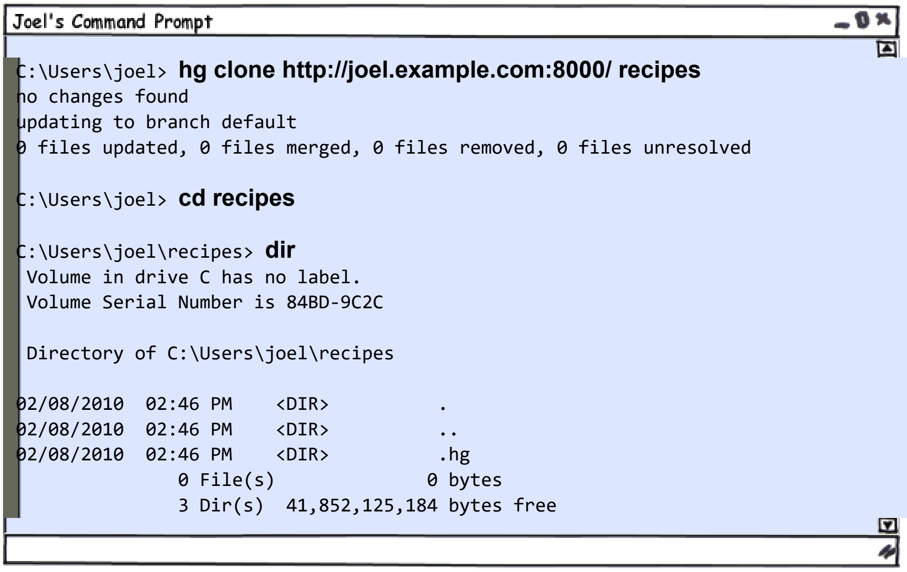
age author description

rev -1: (0) tip

hg clone

make a complete copy of an entire repository

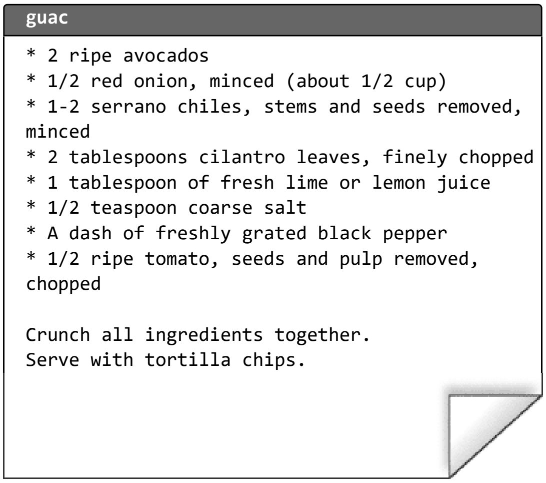
With the central web server running, I can *clone* this repository *from* the server *onto* my own computer for my own use. This repository is empty right now, so I'll just get another empty repository when I clone it.



Joel's Command Prompt

```
C:\Users\joel> hg clone http://joel.example.com:8000/ recipes
o changes found
updating to branch default
0 files updated, 0 files merged, 0 files removed, 0 files unresolved

C:\Users\joel> cd recipes

C:\Users\joel\recipes> dir
 Volume in drive C has no label.
 Volume Serial Number is 84BD-9C2C

 Directory of C:\Users\joel\recipes

02/08/2010  02:46 PM    <DIR>          .
02/08/2010  02:46 PM    <DIR>          ..
02/08/2010  02:46 PM    <DIR>          .hg
               0 File(s)              0 bytes
               3 Dir(s)  41,852,125,184 bytes free
```

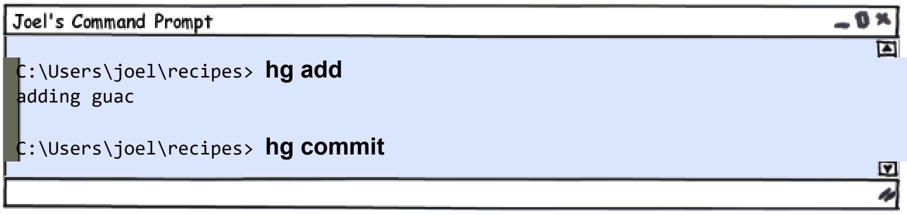
Now I'll create a file called **guac** with my famous guacamole recipe.



guac

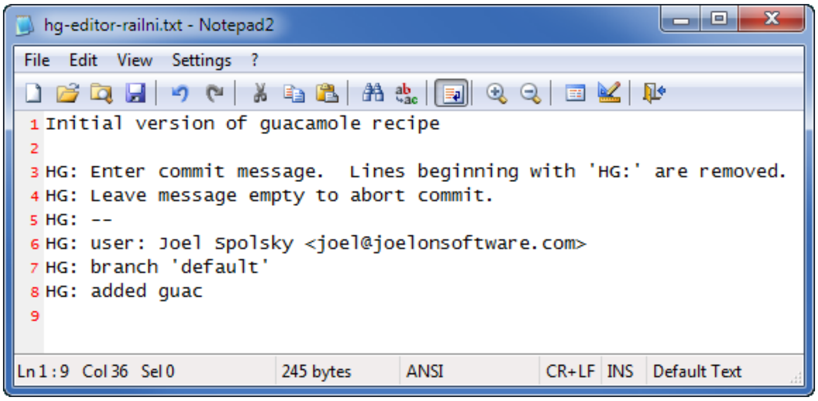
```
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 serrano chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Crunch all ingredients together.
Serve with tortilla chips.
```

I'll add this file, and commit it as the first official version:



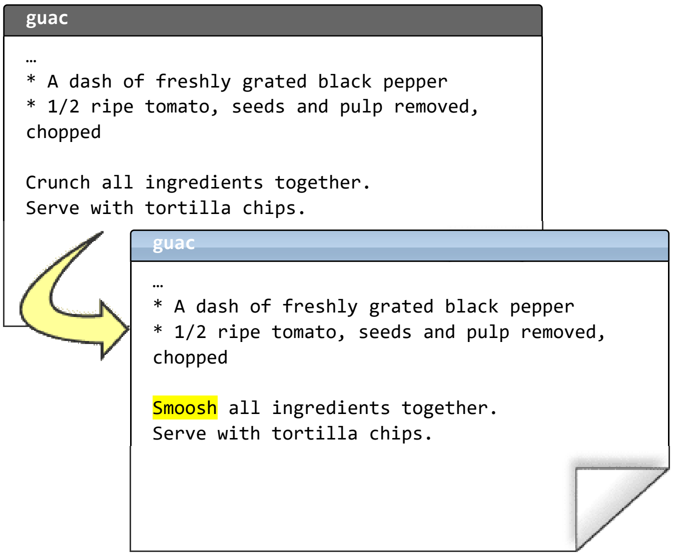
```
C:\Users\joel\recipes> hg add
adding guac

C:\Users\joel\recipes> hg commit
```

I'll provide a commit comment:



```
Initial version of guacamole recipe

HG: Enter commit message.  Lines beginning with 'HG:' are removed.
HG: Leave message empty to abort commit.
HG: --
HG: user: Joel Spolsky <joel@joelonsoftware.com>
HG: branch 'default'
HG: added guac

```

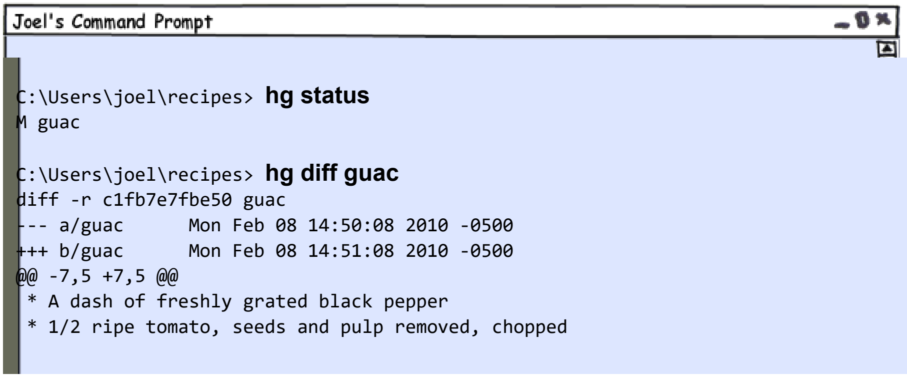
I'm just going to quickly edit this file and make one small change, so that we have a little bit of history in the repository.



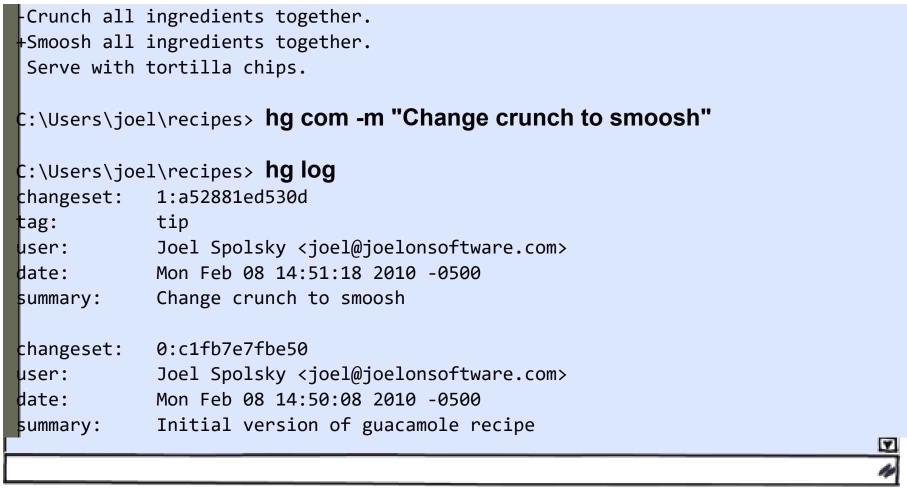
```
…
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Crunch all ingredients together.
Serve with tortilla chips.
```

```
…
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Smoosh all ingredients together.
Serve with tortilla chips.
```

And now to commit that change:



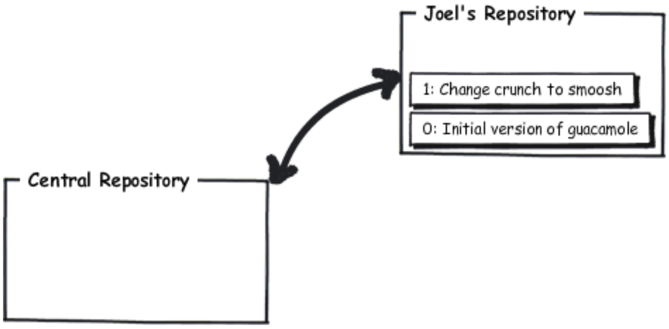
```
C:\Users\joel\recipes> hg status
M guac

C:\Users\joel\recipes> hg diff guac
diff -r c1fb7e7fbe50 guac
--- a/guac      Mon Feb 08 14:50:08 2010 -0500
+++ b/guac      Mon Feb 08 14:51:08 2010 -0500
@@ -7,5 +7,5 @@
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
```



```
-Crunch all ingredients together.
+Smoosh all ingredients together.
 Serve with tortilla chips.

C:\Users\joel\recipes> hg com -m "Change crunch to smoosh"

C:\Users\joel\recipes> hg log
changeset:   1:a52881ed530d
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:51:18 2010 -0500
summary:     Change crunch to smoosh

changeset:   0:c1fb7e7fbe50
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:50:08 2010 -0500
summary:     Initial version of guacamole recipe
```

Notice that when I committed that time, I used the **-m** argument, which I haven't done before. That's just a way to provide the commit message on the command line, without using an editor.

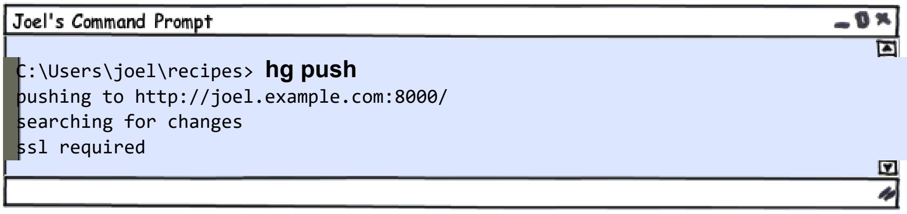
OK, where are we? So far, I've got a central repository, and my clone of it. I've made two changes and committed them, but those changes are only in my clone—they're not in the central repository yet. So the world looks like this:



hg push

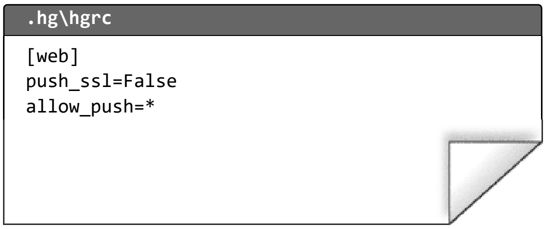
push new changes from this repository into another

Now I'm going to use the **hg push** command, which will push my changes from my repository into the central repository:



```
C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
ssl required
```

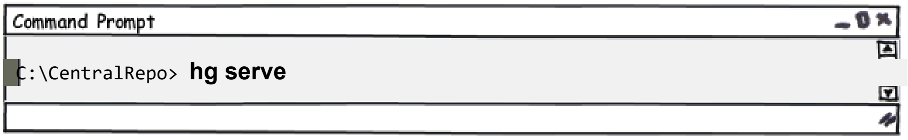
Oh great. It figures that wouldn't work. I neglected to think about the security implications of just running a random web server and allowing anybody in the world to push their stupid changes into it. Bear with me for a moment; I'm going to configure that server to allow anybody in the world to do anything they want to it. This can be done by editing the file .hg\hgrc on the server:



```
[web]
push_ssl=False
allow_push=*
```

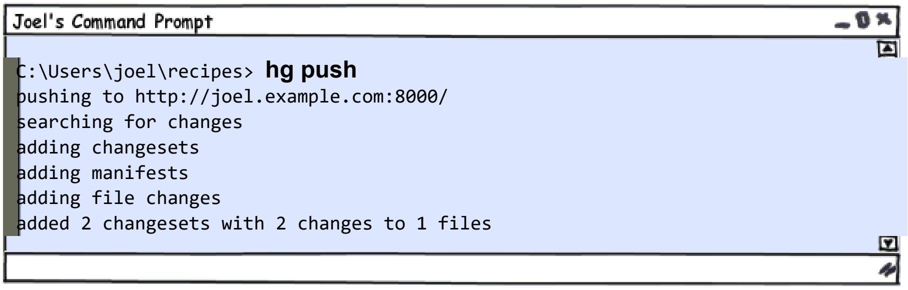
Needless to say, this is rather unsafe, but if you’re on a nice protected LAN at work and there’s a good firewall and you trust everybody on your LAN, this is reasonably OK. Otherwise, you’ll want to read the advanced chapters on security.

OK, time to fire up the server again:



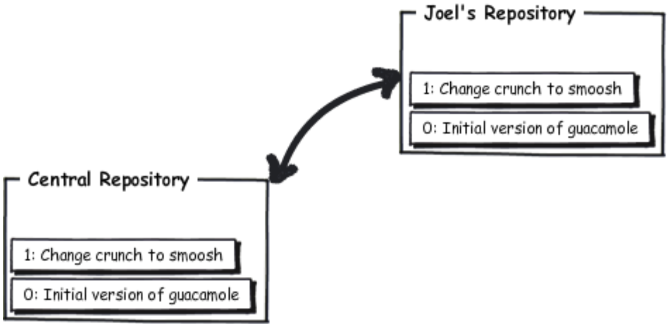
```
C:\CentralRepo> hg serve
```

And now I should be able to push into it:



```
C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
```

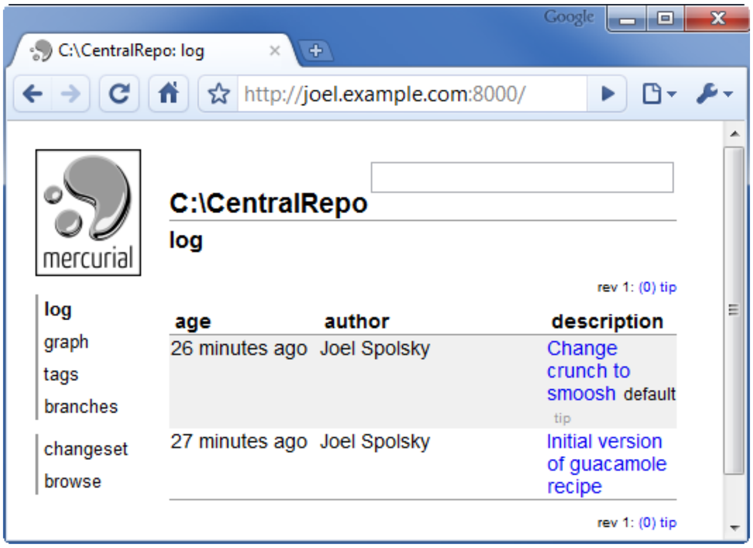
Yay! Now the world looks like this:


Joel's Repository
1: Change crunch to smoosh
0: Initial version of guacamole
Central Repository
1: Change crunch to smoosh
0: Initial version of guacamole

I know what you’re thinking. You’re thinking, “Gosh, Joel, that’s strange. Why do these repositories contain *changes* as opposed to *files*? Where’s that **guac** file?”

Yeah, it’s weird. But that’s the way distributed version control works. Repositories just contain big stacks of changes. Imagine that a change is like one of those clear transparency sheets. When you have a bunch of transparency sheets, you stack them on each other in order, with the latest change on the top, and look down from above, and—tada!—you’re looking at the current version of the file. As you peel away transparencies from the top of the pile, you see older and older versions.

We can use our web browser to peek into the central repository now:



C:\CentralRepo

log

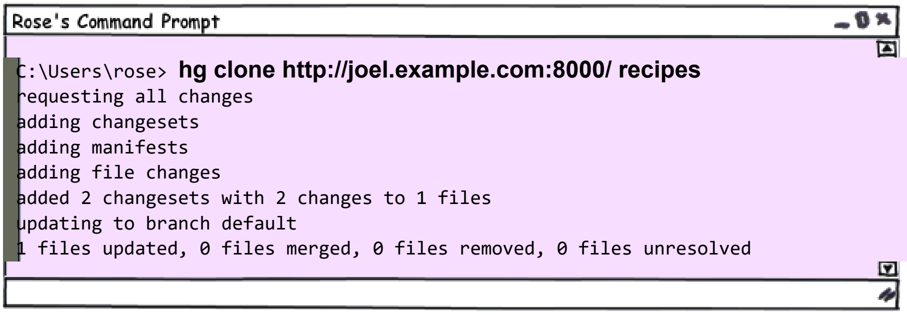
rev 1: (0) tip

age	author	description
26 minutes ago	Joel Spolsky	Change crunch to smoosh default tip
27 minutes ago	Joel Spolsky	Initial version of guacamole recipe

rev 1: (0) tip

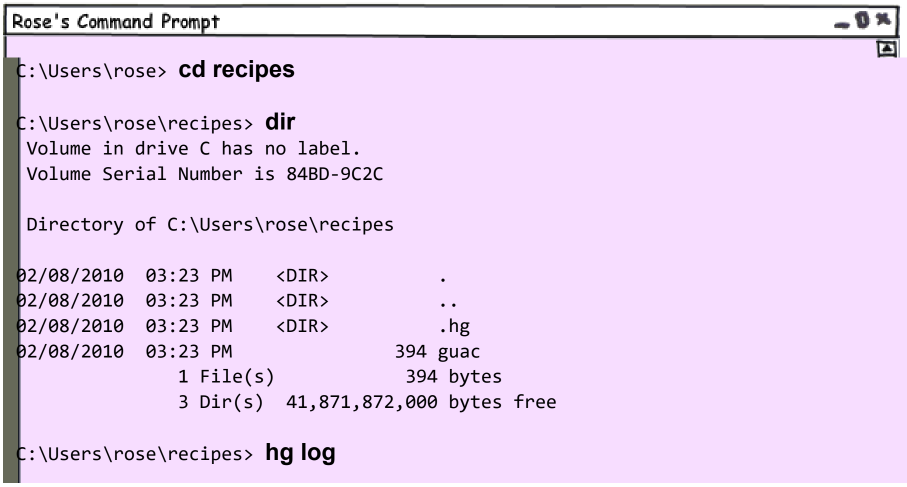
Exactly what you would expect.

Now I want Rose to help me work on the recipe. Rose is on the test team. Everyone agrees that she reminds them of those middle-aged ladies you see in Vegas, sitting there slack-jawed for hours, shoveling quarter after quarter into the slot machines, only she's testing software. You can throw her a new version of your code and she'll test it on 23 different Linux distros, one after the other, expressionless, unmoving, pausing only to tell you that there's a dot missing on one of the lower-case I's in the Turkish version on Ubuntu Linux. Rose is a great tester but I swear sometimes she acts like a zombie.

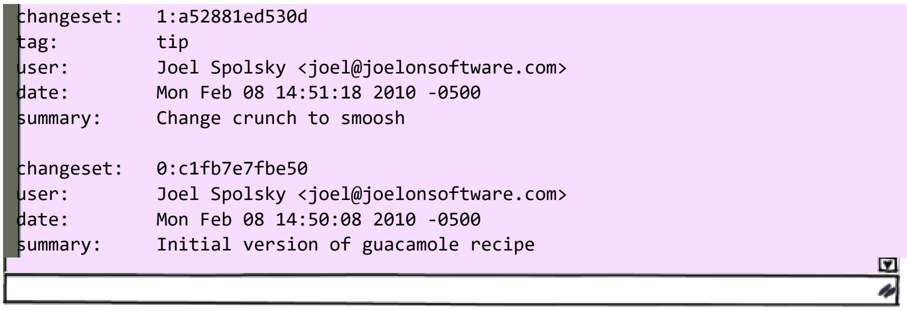


```
C:\Users\rose> hg clone http://joel.example.com:8000/ recipes
requesting all changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
updating to branch default
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Rose used the **hg clone** command to get her own, complete copy of the repository. **hg clone** takes two arguments: the URL of the repository and the name of the directory where you want it cloned. She made her own **recipes** folder.



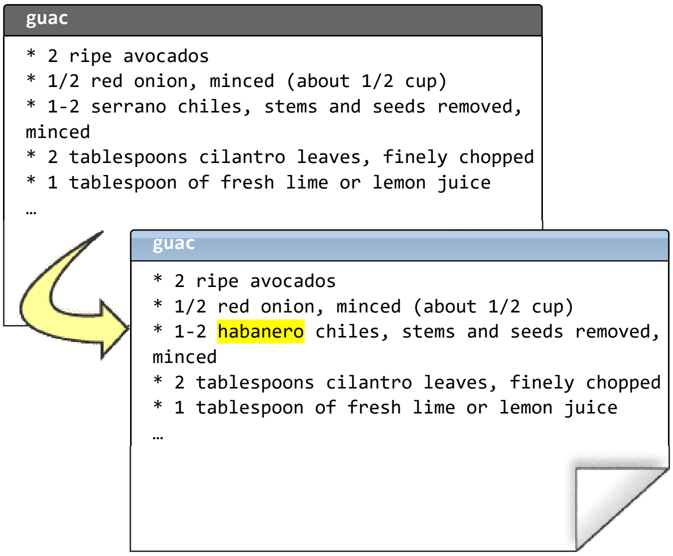
```
C:\Users\rose> cd recipes

C:\Users\rose\recipes> dir
 Volume in drive C has no label.
 Volume Serial Number is 84BD-9C2C

 Directory of C:\Users\rose\recipes

02/08/2010  03:23 PM    <DIR>          .
02/08/2010  03:23 PM    <DIR>          ..
02/08/2010  03:23 PM    <DIR>          .hg
02/08/2010  03:23 PM               394 guac
               1 File(s)            394 bytes
               3 Dir(s)  41,871,872,000 bytes free

C:\Users\rose\recipes> hg log
```



```
changeset:   1:a52881ed530d
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:51:18 2010 -0500
summary:     Change crunch to smoosh

changeset:   0:c1fb7e7fbe50
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:50:08 2010 -0500
summary:     Initial version of guacamole recipe
```

Notice that when she types **hg log** she sees the whole history. She has actually downloaded the entire repository, with its complete history of everything that happened.

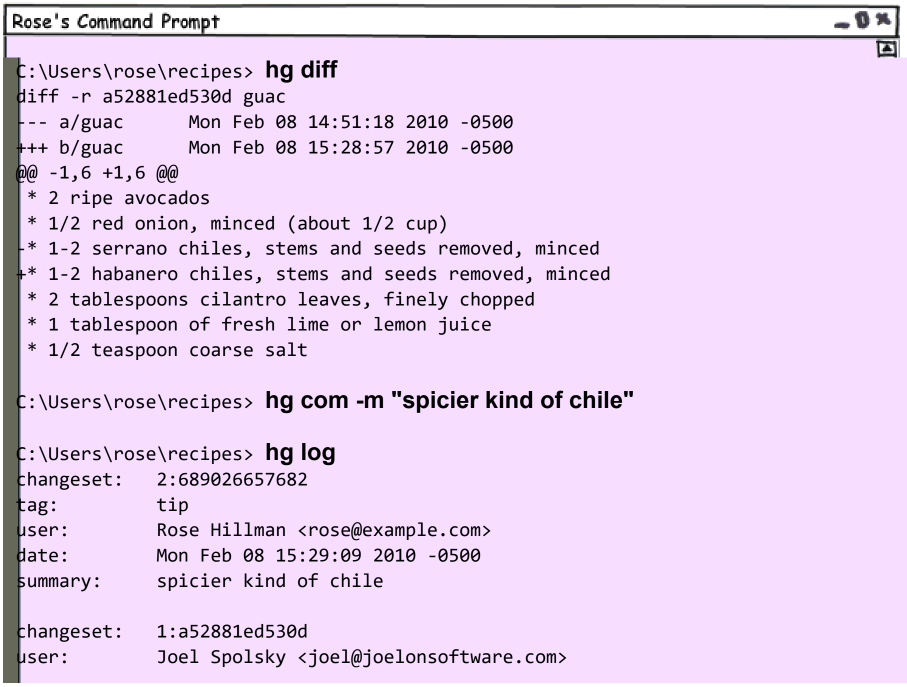
Rose is going to make a change, and check it in:



```
guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 serrano chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
…
```

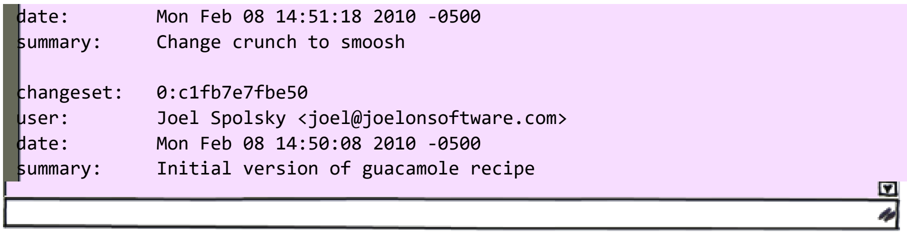
```
guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
…
```

Now she commits it. Notice that she can do this even if the server is not running: the commit entirely happens on her machine.



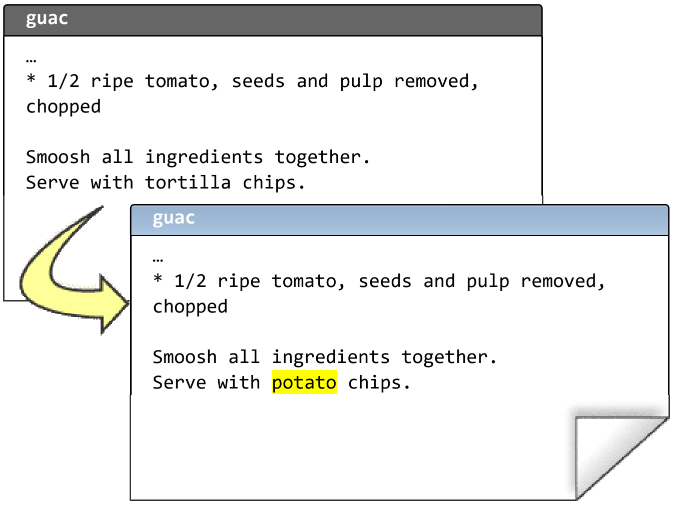
Rose's Command Prompt

```
C:\Users\rose\recipes> hg diff
diff -r a52881ed530d guac
--- a/guac      Mon Feb 08 14:51:18 2010 -0500
+++ b/guac      Mon Feb 08 15:28:57 2010 -0500
@@ -1,6 +1,6 @@
 * 2 ripe avocados
 * 1/2 red onion, minced (about 1/2 cup)
-* 1-2 serrano chiles, stems and seeds removed, minced
+* 1-2 habanero chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped
 * 1 tablespoon of fresh lime or lemon juice
 * 1/2 teaspoon coarse salt

C:\Users\rose\recipes> hg com -m "spicier kind of chile"

C:\Users\rose\recipes> hg log
changeset:   2:689026657682
tag:         tip
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile

changeset:   1:a52881ed530d
user:        Joel Spolsky <joel@joelonsoftware.com>
```



```
date:        Mon Feb 08 14:51:18 2010 -0500
summary:     Change crunch to smoosh

changeset:   0:c1fb7e7fbe50
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:50:08 2010 -0500
summary:     Initial version of guacamole recipe
```

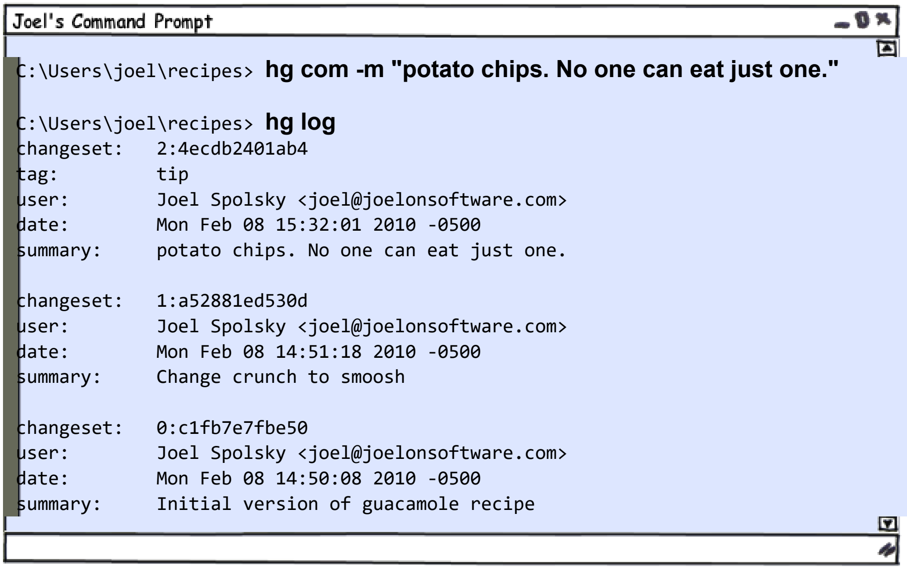
While Rose was making her change, I can make a change at the same time.



```
…
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Smoosh all ingredients together.
Serve with tortilla chips.
```

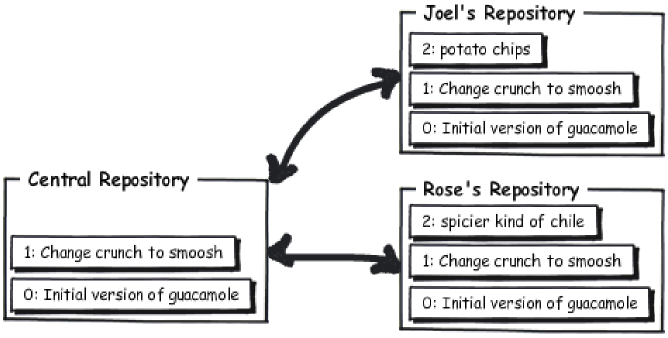
```
…
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Smoosh all ingredients together.
Serve with potato chips.
```

After I check that in, you'll see that my log shows something different as changeset #2 than Rose's log.



```
C:\Users\joel\recipes> hg com -m "potato chips. No one can eat just one."

C:\Users\joel\recipes> hg log
changeset:   2:4ecdb2401ab4
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 15:32:01 2010 -0500
summary:     potato chips. No one can eat just one.

changeset:   1:a52881ed530d
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:51:18 2010 -0500
summary:     Change crunch to smoosh

changeset:   0:c1fb7e7fbe50
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:50:08 2010 -0500
summary:     Initial version of guacamole recipe
```

Our histories are starting to diverge.

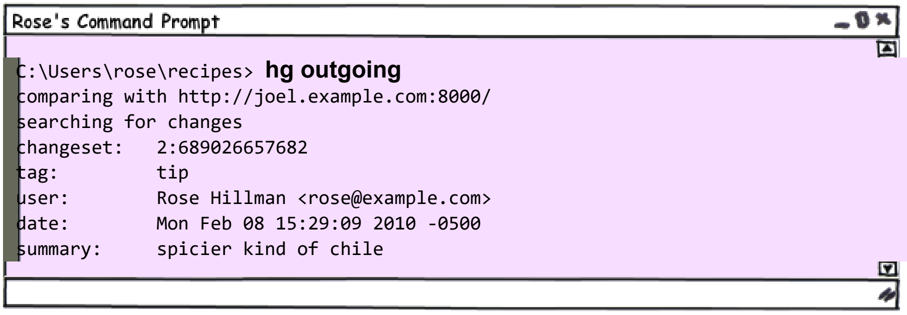

Joel's Repository
2: potato chips
1: Change crunch to smoosh
0: Initial version of guacamole
Central Repository
1: Change crunch to smoosh
0: Initial version of guacamole
Rose's Repository
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole

Don’t worry… in a minute we’ll see how to bring these diverging changes back together into one delicious habanero-based potato chip dip.

hg outgoing

list changes in current repository waiting to be pushed

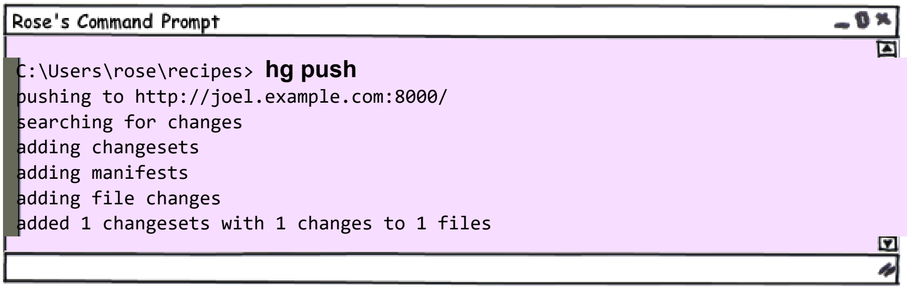
Rose can continue to work, disconnected, making as many changes as she wants, and either committing them, or reverting them, in her own repository. At some point, though, she’s going to want to share all the changes she’s been committing with the outside world. She can type **hg outgoing** which will show a list of changes that are waiting to be sent up to the central repository. These are the changes that **hg push** would send, if she were to **hg push**.



```
C:\Users\rose\recipes> hg outgoing
comparing with http://joel.example.com:8000/
searching for changes
changeset:   2:689026657682
tag:         tip
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile
```

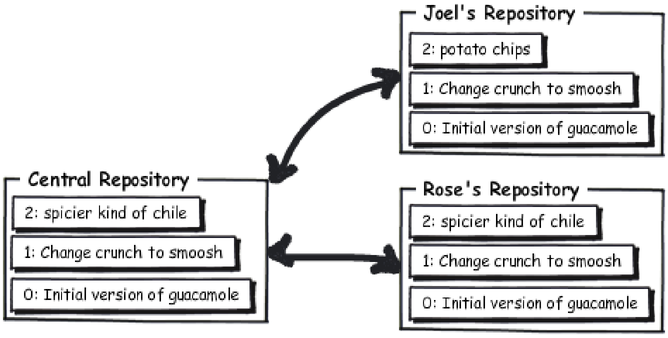
Think of **hg outgoing** like this: it simply lists any changes in the current repository that aren’t in the central repository.

OK, so Rose pushes her changes.

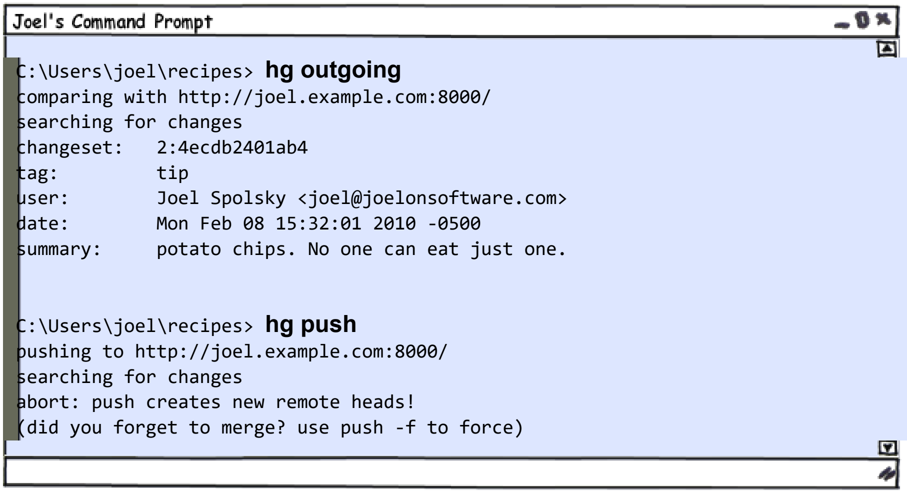


```
C:\Users\rose\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

And the world looks like this:


Joel's Repository
2: potato chips
1: Change crunch to smoosh
0: Initial version of guacamole
Central Repository
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole
Rose's Repository
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole

When I get back from my fourth latte break of the day, I'm ready to push my potato-chip change, too.



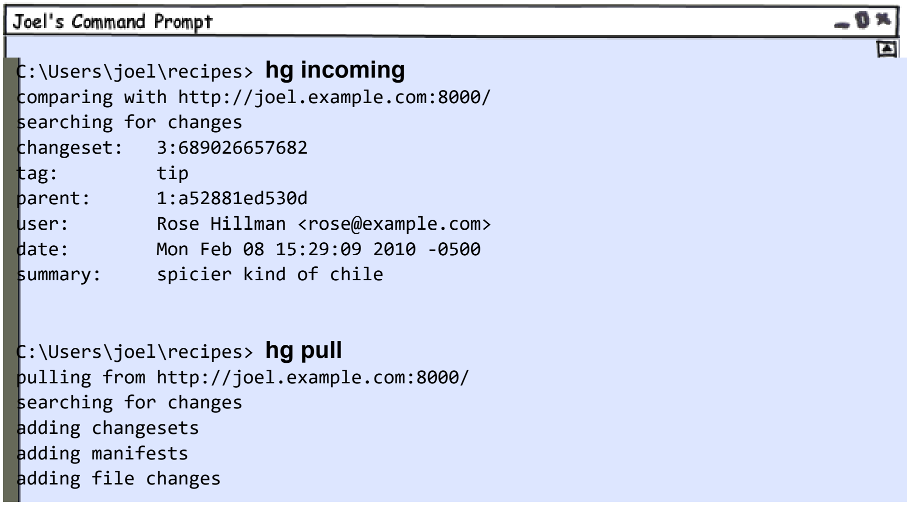
```
C:\Users\joel\recipes> hg outgoing
comparing with http://joel.example.com:8000/
searching for changes
changeset:   2:4ecdb2401ab4
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 15:32:01 2010 -0500
summary:     potato chips. No one can eat just one.


C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
abort: push creates new remote heads!
(did you forget to merge? use push -f to force)
```

Ahhh!! Failure! By the way… that message you see there? The one that says **use push -f to force?** That's *terrible* advice. Never, ever, EVER use **push -f** to force. You will regret it. Trust me for now.

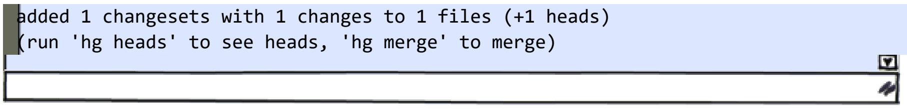
The reason Rose's push succeeded while mine failed is because potato chips do not go well with guacamole. Just kidding! I wanted to see if you were awake, there.

The push failed because we both made changes, and so they need to be merged somehow, and Mercurial knows it.

The first thing I'm going to do is get all those changes that are in the central repository that I don't have yet, so I can merge them.

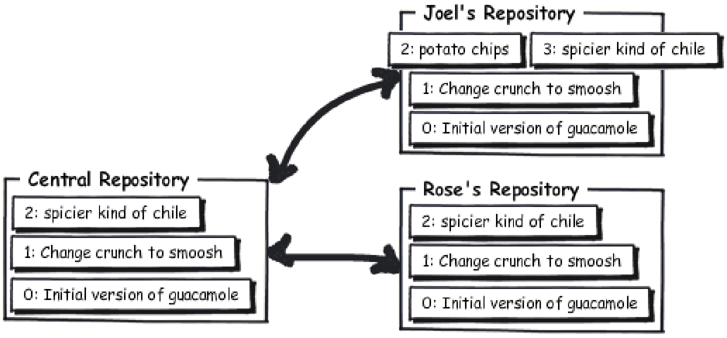


```
C:\Users\joel\recipes> hg incoming
comparing with http://joel.example.com:8000/
searching for changes
changeset:   3:689026657682
tag:         tip
parent:      1:a52881ed530d
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile


C:\Users\joel\recipes> hg pull
pulling from http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
```

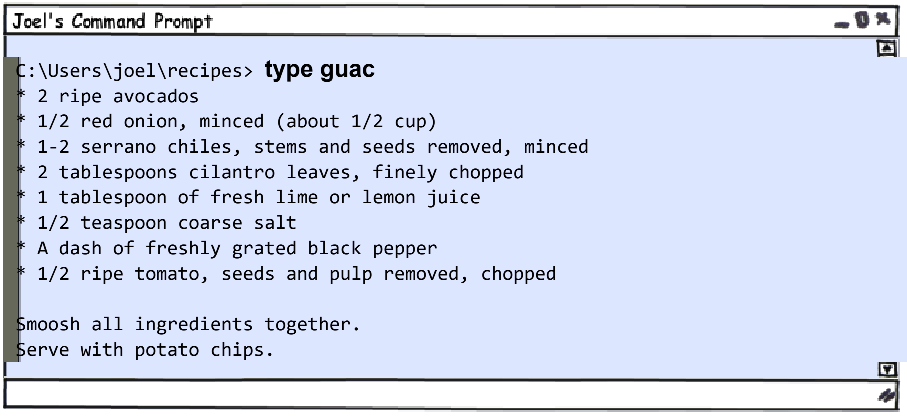


```
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)
```

There's some gibberish there about +1 heads. That's because my repository, which used to just have three changes neatly stacked, is now a two-headed monster, with two different changes stacked precariously on top like this:

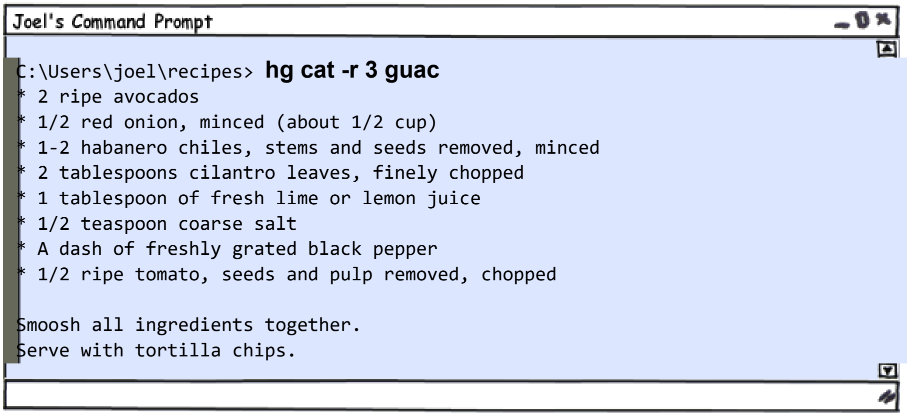

Joel's Repository
2: potato chips
3: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole
Central Repository
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole
Rose's Repository
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole

I've got both versions in my repository now… I've got my version:



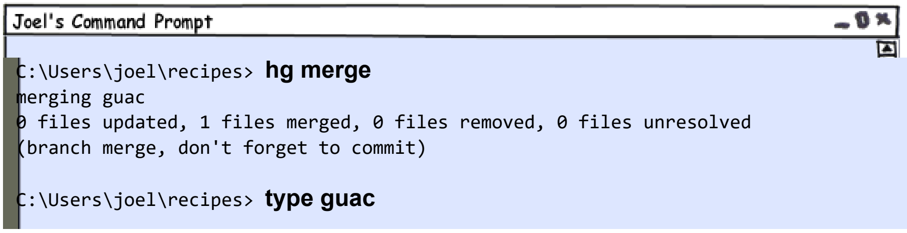
```
C:\Users\joel\recipes> type guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 serrano chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with potato chips.
```

And I've got Rose's version:

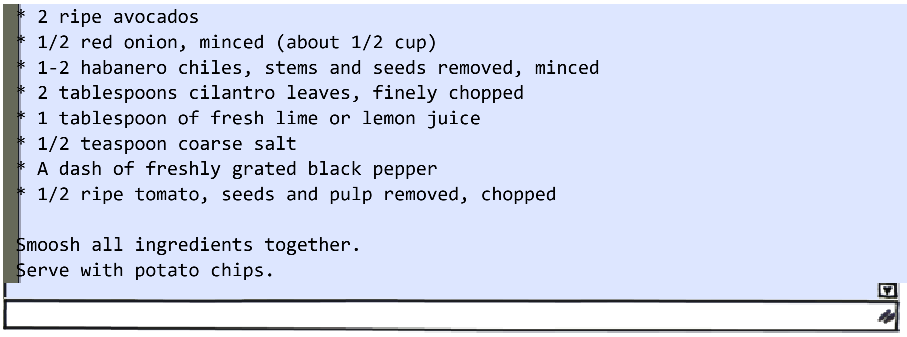


```
C:\Users\joel\recipes> hg cat -r 3 guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with tortilla chips.
```

And it's up to me to merge them. Luckily, this is easy.



```
C:\Users\joel\recipes> hg merge
merging guac
0 files updated, 1 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\joel\recipes> type guac
```



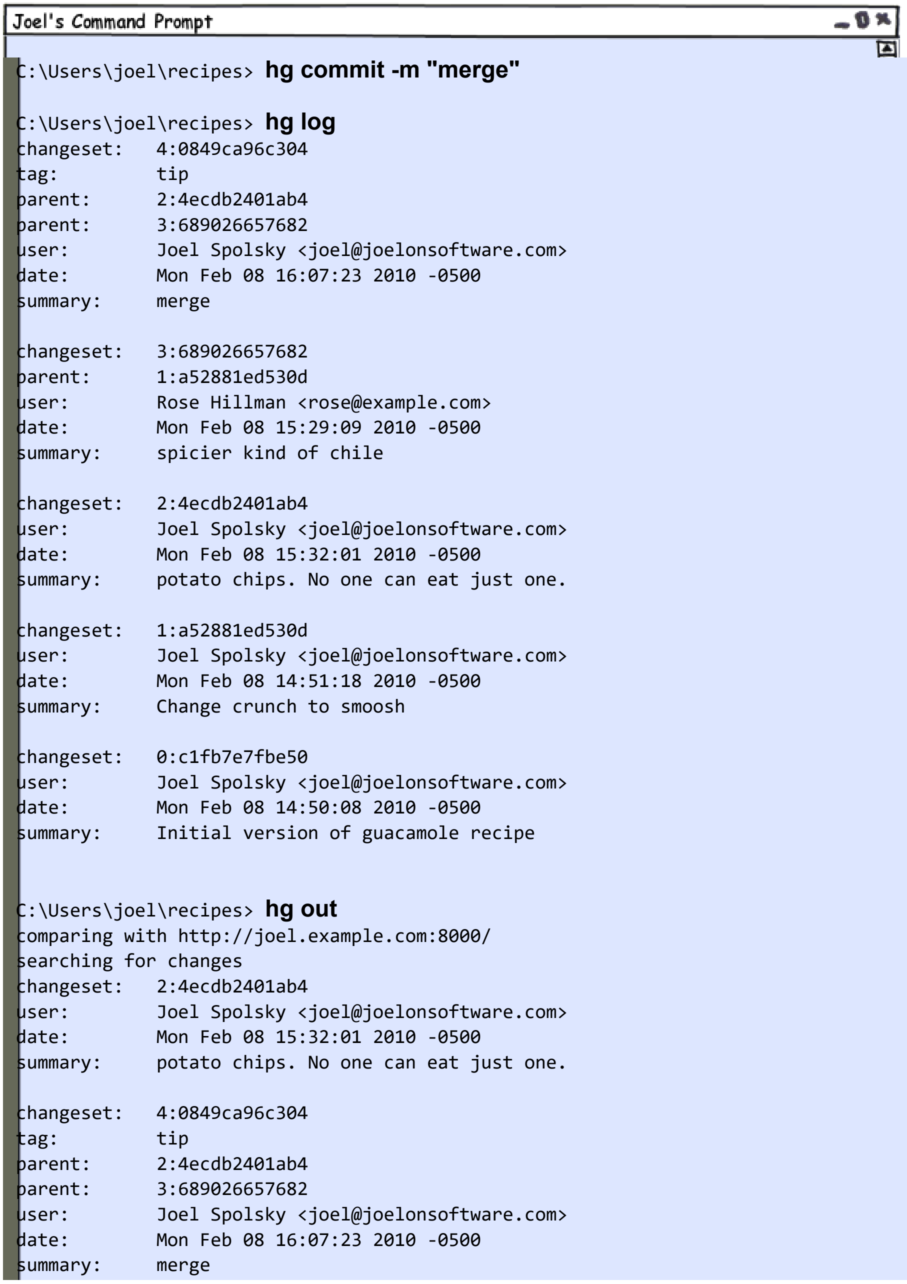
```
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with potato chips.
```

Look! The **hg merge** command took my two heads and combined them together. In this case, since we both edited different parts of the file, there was no conflict at all and the merge went off without a hitch.

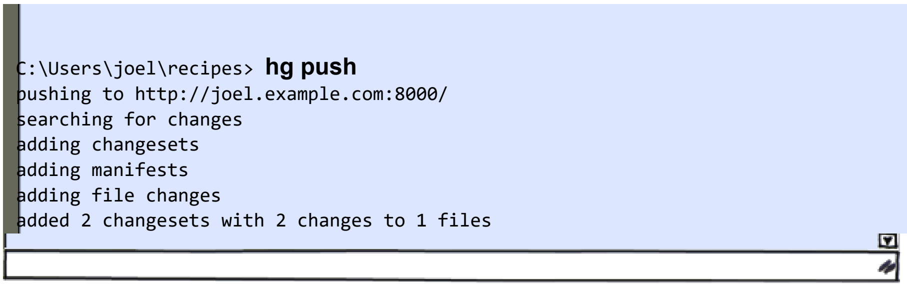
hg merge

merge two heads

I still have to commit. This is important. If the merge failed, I could always revert and try again. Since the merge was successful, I'm going to commit it. Then I'll be able to push my changes to the central repository.

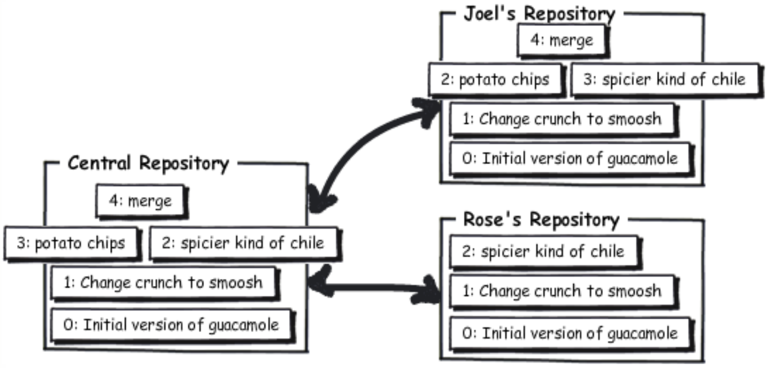


```
C:\Users\joel\recipes> hg commit -m "merge"

C:\Users\joel\recipes> hg log
changeset:   4:0849ca96c304
tag:         tip
parent:      2:4ecdb2401ab4
parent:      3:689026657682
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge

changeset:   3:689026657682
parent:      1:a52881ed530d
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile

changeset:   2:4ecdb2401ab4
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 15:32:01 2010 -0500
summary:     potato chips. No one can eat just one.

changeset:   1:a52881ed530d
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:51:18 2010 -0500
summary:     Change crunch to smoosh

changeset:   0:c1fb7e7fbe50
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:50:08 2010 -0500
summary:     Initial version of guacamole recipe


C:\Users\joel\recipes> hg out
comparing with http://joel.example.com:8000/
searching for changes
changeset:   2:4ecdb2401ab4
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 15:32:01 2010 -0500
summary:     potato chips. No one can eat just one.

changeset:   4:0849ca96c304
tag:         tip
parent:      2:4ecdb2401ab4
parent:      3:689026657682
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge
```



```
C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
```

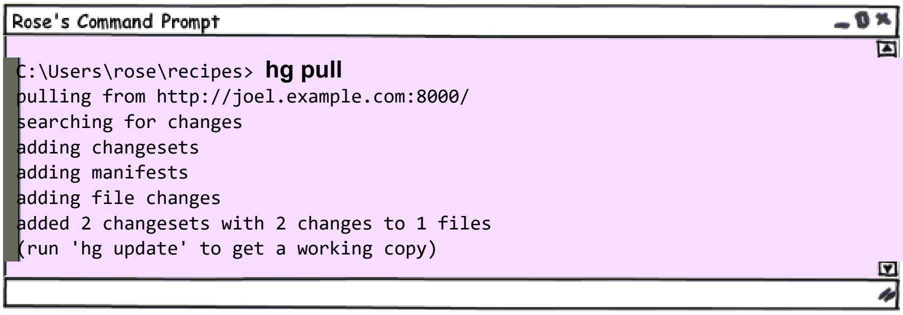
And now the central repository has the same thing as I do:


Joel's Repository
4: merge
2: potato chips
3: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole
Central Repository
4: merge
3: potato chips
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole
Rose's Repository
2: spicier kind of chile
1: Change crunch to smoosh
0: Initial version of guacamole

OK, I have Rose's changes, and my changes, but Rose doesn't have my changes yet.

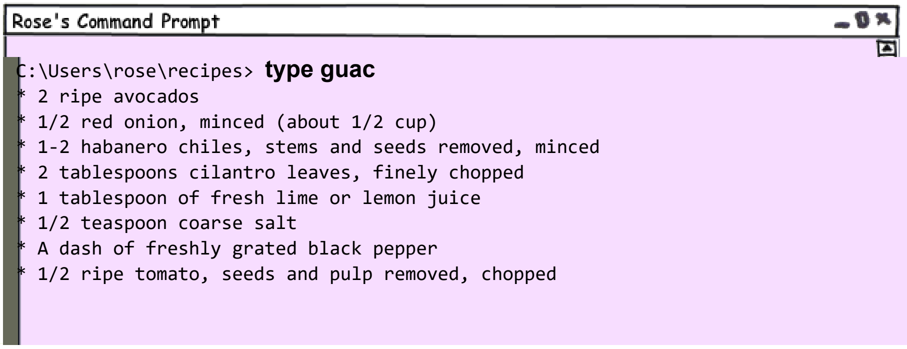
One thing I forgot to tell you about Rose. She's a doctor. Yep. A medical doctor. Isn't that weird? She was a hotshot pediatrician at Mt. Sinai, probably earning five times as much as this crappy joint pays its testers. Nobody really knows why she left the field of medicine. The other testers think something tragic happened. She had a family, once, too; there's a picture of a cute ten year old on her desk, but now she lives alone, and we don't want to pry.

Rose needs to pull the latest, incoming stuff from the repository to get it.

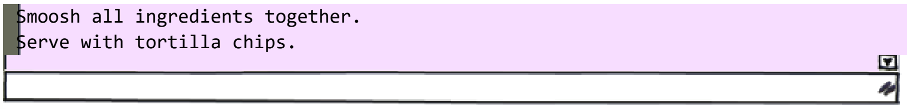


```
C:\Users\rose\recipes> hg pull
pulling from http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
(run 'hg update' to get a working copy)
```

Got it. Now, you may find this a bit odd, but even though Rose pulled those new changes into her repository, *they're not in her working directory yet*.



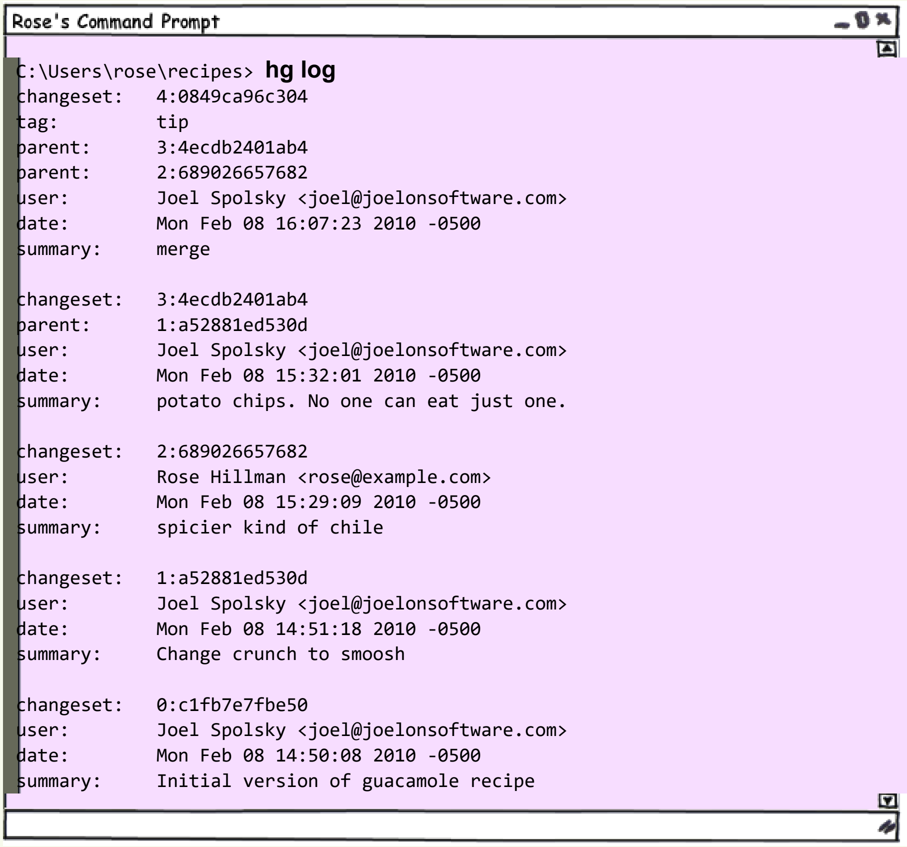
```
C:\Users\rose\recipes> type guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped
```



```
Smoosh all ingredients together.
Serve with tortilla chips.
```

See that? She’s still working with Tortilla chips. Tortilla chips!

She *does* have my new changes somewhere in her repository…



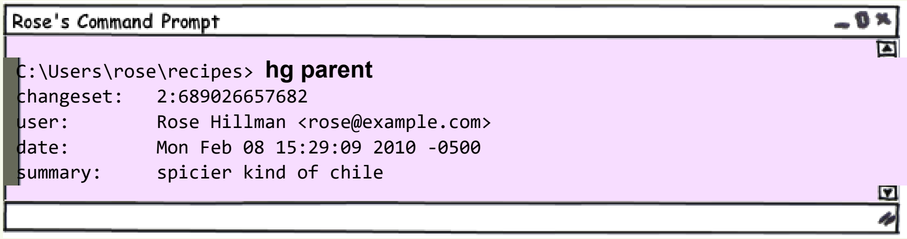
Rose's Command Prompt

```
C:\Users\rose\recipes> hg log
changeset:   4:0849ca96c304
tag:         tip
parent:      3:4ecdb2401ab4
parent:      2:689026657682
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge

changeset:   3:4ecdb2401ab4
parent:      1:a52881ed530d
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 15:32:01 2010 -0500
summary:     potato chips. No one can eat just one.

changeset:   2:689026657682
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile

changeset:   1:a52881ed530d
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:51:18 2010 -0500
summary:     Change crunch to smoosh

changeset:   0:c1fb7e7fbe50
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 14:50:08 2010 -0500
summary:     Initial version of guacamole recipe
```

hg parent

show the changeset that’s in the working directory

They’re just not in her working directory. That’s because she’s still working off of changeset #2. You can see this with the “parent” command:

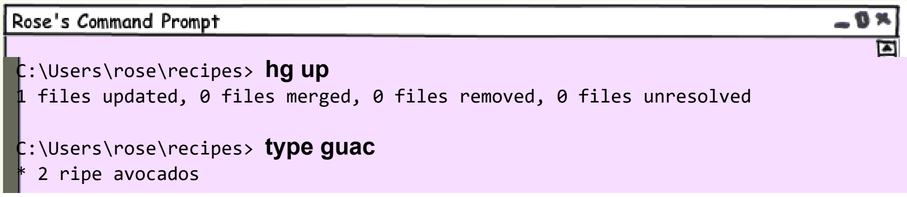


Rose's Command Prompt

```
C:\Users\rose\recipes> hg parent
changeset:   2:689026657682
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile
```

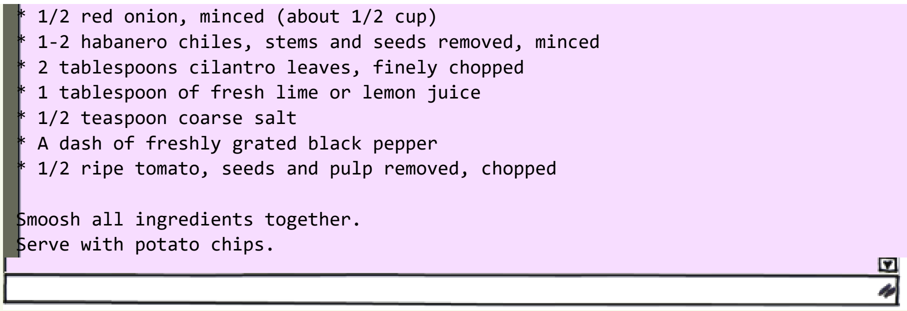
Mercurial is being nice to us. It’s always safe to pull; all it does is get us the latest changes that other people have been making. We can switch to working with them later, at our own convenience.

Remember that the **hg up** command with no arguments will change the working directory to the *tip* (the absolute TOP changeset), in this case number 4:



Rose's Command Prompt

```
C:\Users\rose\recipes> hg up
1 files updated, 0 files merged, 0 files removed, 0 files unresolved

C:\Users\rose\recipes> type guac
* 2 ripe avocados
```



```
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with potato chips.
```

And now, she’s looking at the latest version with everybody’s changes.

When you’re working on a team, your workflow is going to look a lot like this:

1. If you haven’t done so in a while, get the latest version that everyone else is working off of:
   - hg pull
   - hg up
2. Make some changes
3. Commit them (locally)
4. Repeat steps 2-3 until you’ve got some nice code that you’re willing to inflict on everyone else
5. When you’re ready to share:
   - hg pull to get everyone else’s changes (if there are any)
   - hg merge to merge them into yours
   - test! to make sure the merge didn’t screw anything up
   - hg commit (the merge)
   - hg push

Test yourself

Here are the things you should know how to do after reading this tutorial:

1. Set up a central repository and let team members clone off of it
2. Push changes into the central repository
3. Pull changes from the central repository
4. Merge changes from different contributors

« Home

Next, we talk about how to fix ~~misstakes~~ mistakes.

Next »

Any questions?

If you have any questions about the material in this tutorial, no matter how newbie, ask them at the Kiln Knowledge Exchange.

About the author.

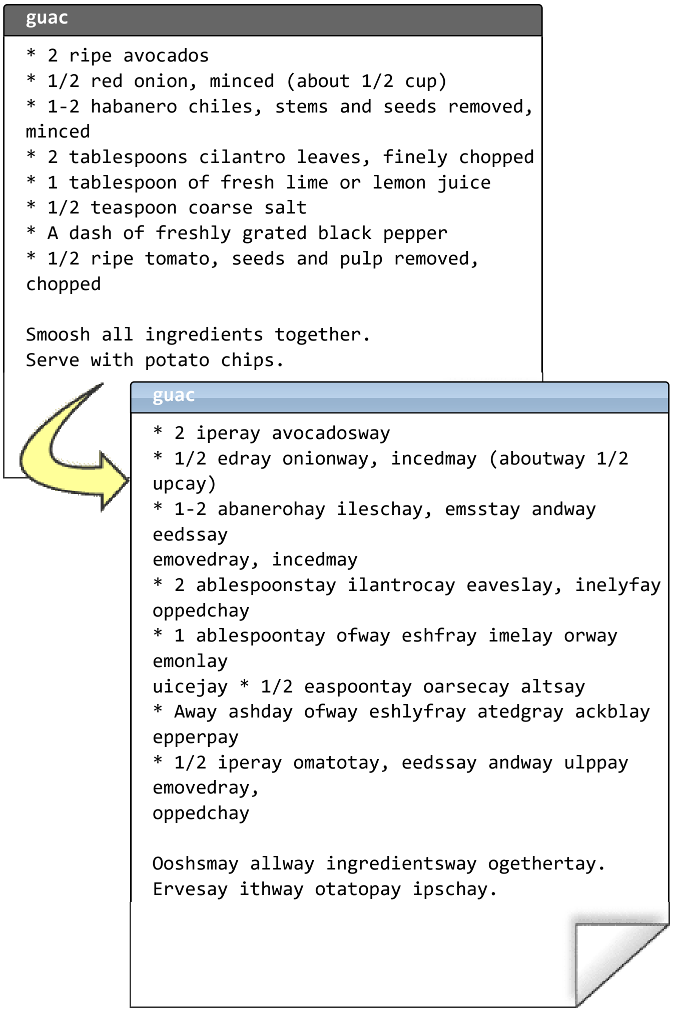
Joel Spolsky is the founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they’re delighted. Fog Creek makes FogBugz, Kiln, and Fog Creek Copilot. Joel's blog Joel on Software is read by programmers everywhere.

80
Hg
Init
200.59

One of the biggest benefits of Mercurial is that you can use private clones to try experiments and develop new features… if they don't work out, you can reverse them in a second.

Fixing Goofs

Mercurial lets you experiment freely. Imagine that during the course of normal editing, you get into trouble with your editor and do something catastrophic:



guac

```
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Smoosh all ingredients together.
Serve with potato chips.
```

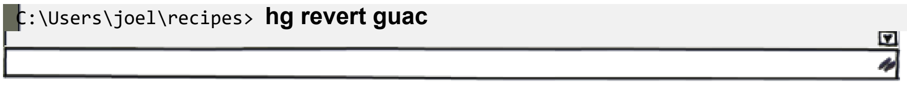
guac

```
* 2 iperay avocadosway
* 1/2 edray onionway, incedmay (aboutway 1/2
upcay)
* 1-2 abanerohay ileschay, emsstay andway
eedssay
emovedray, incedmay
* 2 ablespoonstay ilantrocay eaveslay, inelyfay
oppedchay
* 1 ablespoontay ofway eshfray imelay orway
emonlay
uicejay * 1/2 easpoontay oarsecay altsay
* Away ashday ofway eshlyfray atedgray ackblay
epperpay
* 1/2 iperay omatotay, eedssay andway ulppay
emovedray,
oppedchay

Ooshsmay allway ingredientsway ogethertay.
Ervesay ithway otatopay ipschay.
```

hg revert

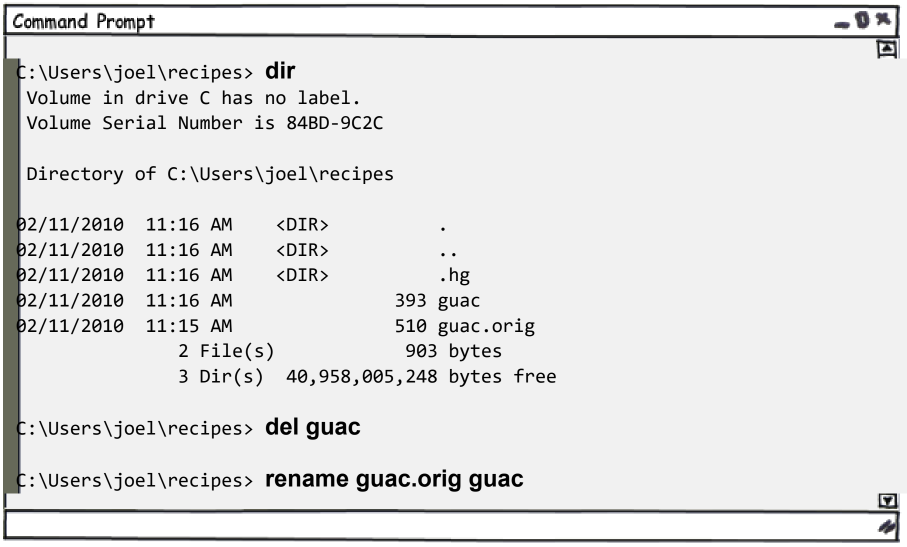
Gotta love emacs. Anyway, all is not lost. The most common way to recover from these things is just to **hg revert** them:

revert changed files back to committed version



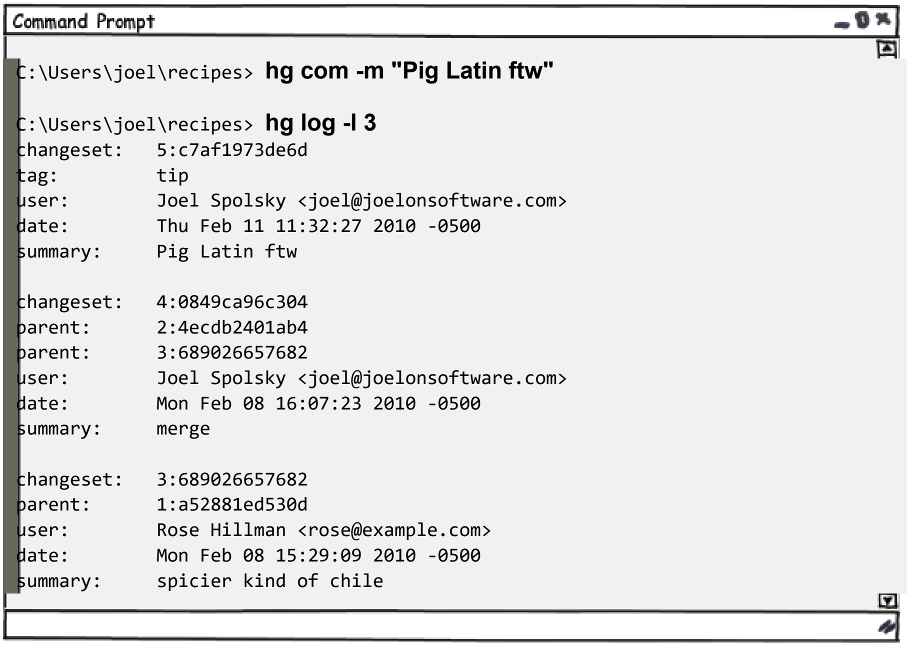
```
C:\Users\joel\recipes> hg revert guac
```

That'll put the files back exactly the way they were at the time of the last commit. Mercurial doesn't like to delete anything, so instead of zapping the Pig Latin recipe, it renamed it:



```
C:\Users\joel\recipes> dir
 Volume in drive C has no label.
 Volume Serial Number is 84BD-9C2C

 Directory of C:\Users\joel\recipes

02/11/2010  11:16 AM    <DIR>          .
02/11/2010  11:16 AM    <DIR>          ..
02/11/2010  11:16 AM    <DIR>          .hg
02/11/2010  11:16 AM               393 guac
02/11/2010  11:15 AM               510 guac.orig
               2 File(s)            903 bytes
               3 Dir(s)  40,958,005,248 bytes free

C:\Users\joel\recipes> del guac

C:\Users\joel\recipes> rename guac.orig guac
```

What if you had gone one step too far and actually committed?

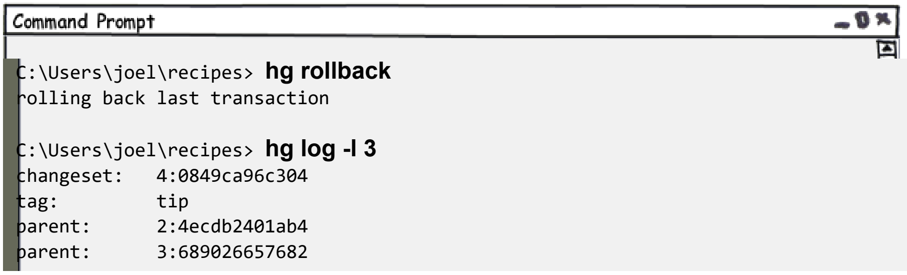


```
C:\Users\joel\recipes> hg com -m "Pig Latin ftw"

C:\Users\joel\recipes> hg log -l 3
changeset:   5:c7af1973de6d
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 11:32:27 2010 -0500
summary:     Pig Latin ftw

changeset:   4:0849ca96c304
parent:      2:4ecdb2401ab4
parent:      3:689026657682
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge

changeset:   3:689026657682
parent:      1:a52881ed530d
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile
```

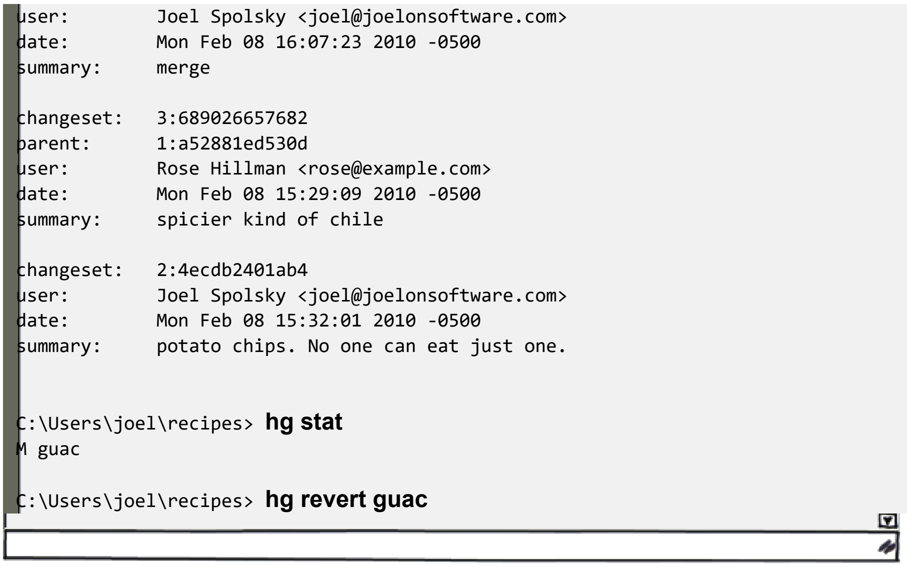
hg rollback

undoes one commit, as long as you haven't pushed it to anyone else.

There's a command called **hg rollback** which will save your skin, but only if you haven't pushed this change to anyone else. It only undoes *one* commit.



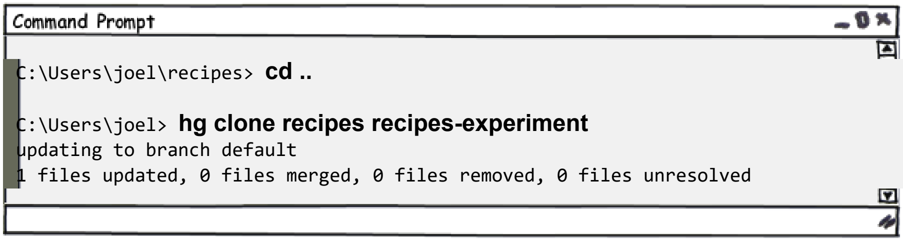
```
C:\Users\joel\recipes> hg rollback
rolling back last transaction

C:\Users\joel\recipes> hg log -l 3
changeset:   4:0849ca96c304
tag:         tip
parent:      2:4ecdb2401ab4
parent:      3:689026657682
```



```
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge

changeset:   3:689026657682
parent:      1:a52881ed530d
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile

changeset:   2:4ecdb2401ab4
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 15:32:01 2010 -0500
summary:     potato chips. No one can eat just one.


C:\Users\joel\recipes> hg stat
M guac

C:\Users\joel\recipes> hg revert guac
```

Imagine you want to do a major experiment on the side. Your boss hired a new designer, Jim, and lately the specs you've been getting from him are just absurd. There's fluorescent green text, nothing lines up (for "artistic" reasons), and the usability is awful. You want to come in one weekend and redo the whole thing, but you're afraid to commit it because you're not really 100% sure that your ideas are better than this nutty graphic designer. Jim is basically smoking a joint from the moment he wakes up until he goes to bed. You don't want to hold that against him, and everybody else thinks that it's nobody's business as long as his designs are good, but really, there's a limit. Right? And his designs aren't good. Plus he's kind of offensive.

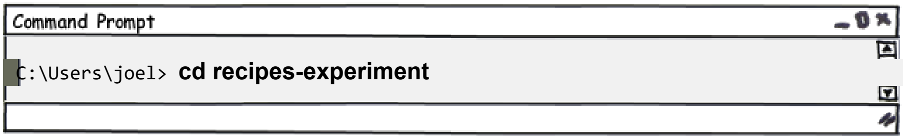
With Mercurial, you can just make an experimental clone of the entire repository:



```
C:\Users\joel\recipes> cd ..

C:\Users\joel> hg clone recipes recipes-experiment
updating to branch default
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

This isn't as inefficient as it seems. Because both **recipes** and **recipes-experiment** share all their history (so far), Mercurial will use a file system trick called "hard links" so that the copy can be created very quickly, without taking up a lot of extra space on disk.

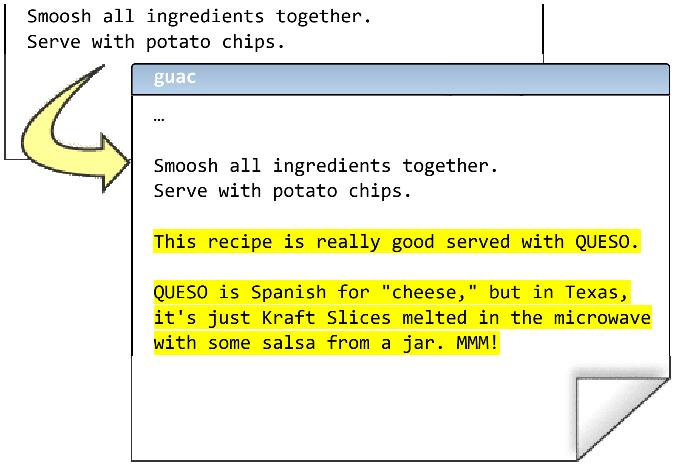
Now we can make a bunch of changes in the experimental branch:



```
C:\Users\joel> cd recipes-experiment
```

Here's my grand guacamole experiment:

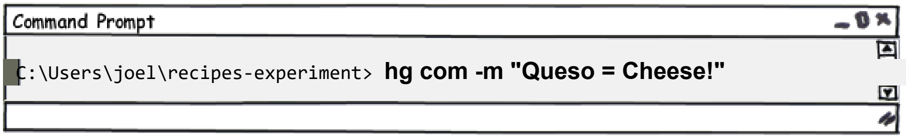
…



```
Smoosh all ingredients together.
Serve with potato chips.
```

```
…

Smoosh all ingredients together.
Serve with potato chips.

This recipe is really good served with QUESO.

QUESO is Spanish for "cheese," but in Texas,
it's just Kraft Slices melted in the microwave
with some salsa from a jar. MMM!
```

Here in the experimental repository, we can commit freely.

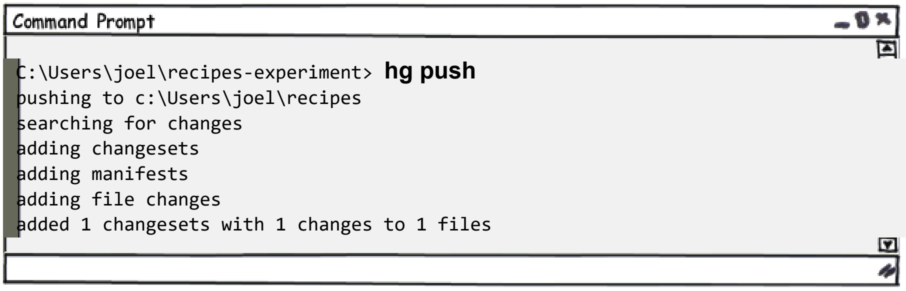


```
C:\Users\joel\recipes-experiment> hg com -m "Queso = Cheese!"
```

You can make changes and work freely, committing whenever you want. That gives you all the power of source control even for your crazy experiment, without infecting anyone else.

If you decide that the experiment was misguided, you can just delete the whole experimental directory. Problem solved. It's gone.

But if it worked, all you have to do is push your new changes:

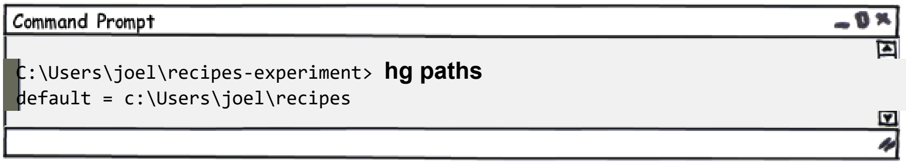


```
C:\Users\joel\recipes-experiment> hg push
pushing to c:\Users\joel\recipes
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

Where did they go?

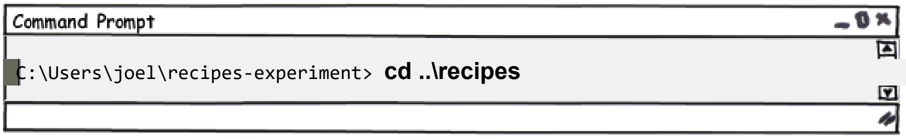
hg paths

shows a list of known remote repositories



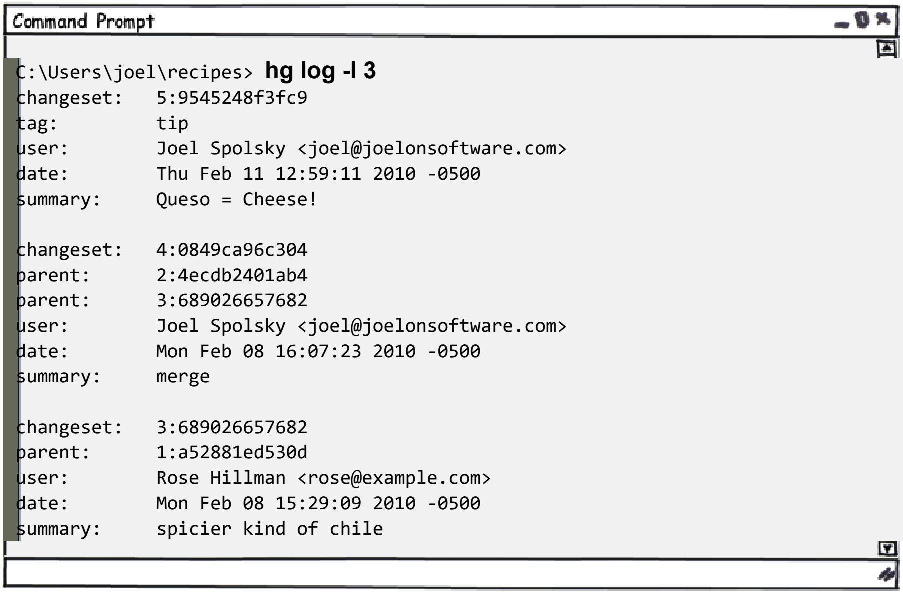
```
C:\Users\joel\recipes-experiment> hg paths
default = c:\Users\joel\recipes
```

The "default" entry there shows you the path to the repository that **hg push** will push changes to, if you don't specify any other repository. Normally, that's the repository that you cloned off of. In this case, it's a local directory, but you could also have a URL there.



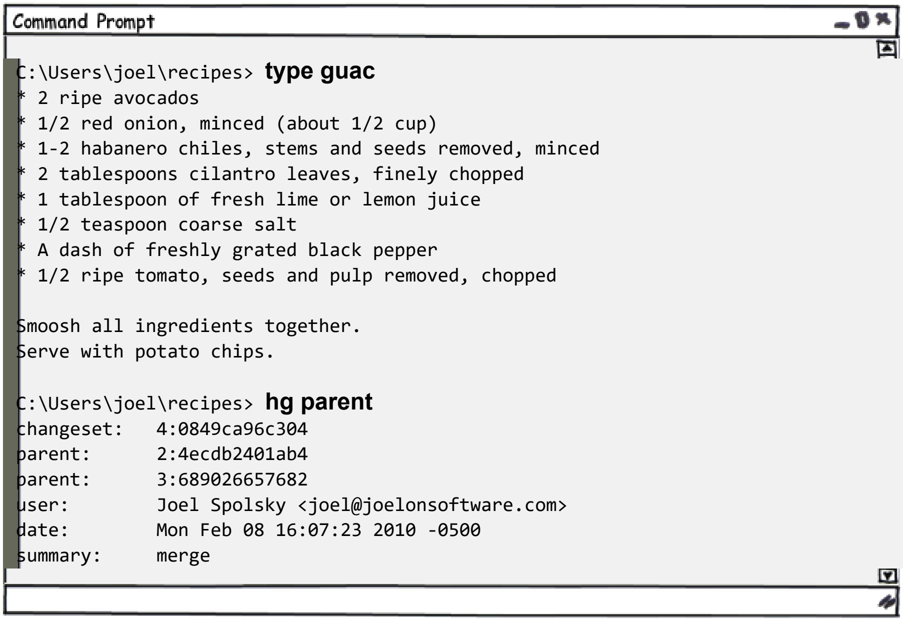
```
C:\Users\joel\recipes-experiment> cd ..\recipes
```

Don’t forget, just because the change has been pushed into this *repository*…



```
C:\Users\joel\recipes> hg log -l 3
changeset:   5:9545248f3fc9
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 12:59:11 2010 -0500
summary:     Queso = Cheese!

changeset:   4:0849ca96c304
parent:      2:4ecdb2401ab4
parent:      3:689026657682
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge

changeset:   3:689026657682
parent:      1:a52881ed530d
user:        Rose Hillman <rose@example.com>
date:        Mon Feb 08 15:29:09 2010 -0500
summary:     spicier kind of chile
```

… doesn’t mean we’re working off that version yet.

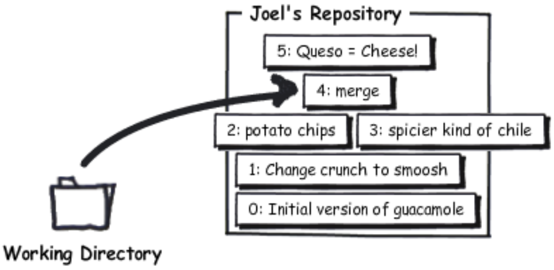


```
C:\Users\joel\recipes> type guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with potato chips.

C:\Users\joel\recipes> hg parent
changeset:   4:0849ca96c304
parent:      2:4ecdb2401ab4
parent:      3:689026657682
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Mon Feb 08 16:07:23 2010 -0500
summary:     merge
```

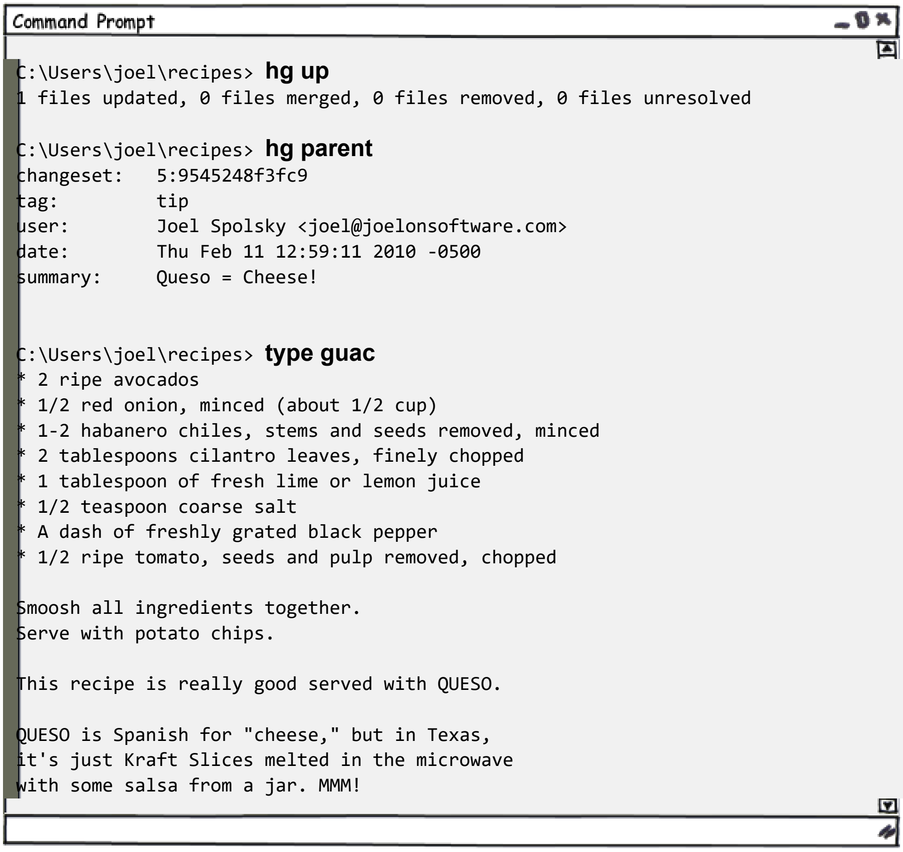
hg parent

shows which changeset(s) you’re working off of

See? That “Queso” stuff is in changeset 5. But my main repository was working off of changeset 4, and just because someone pushed new changes into the *repository*, doesn’t mean they’ve showed up in my working directory yet, so I’m still working off of changeset 4.



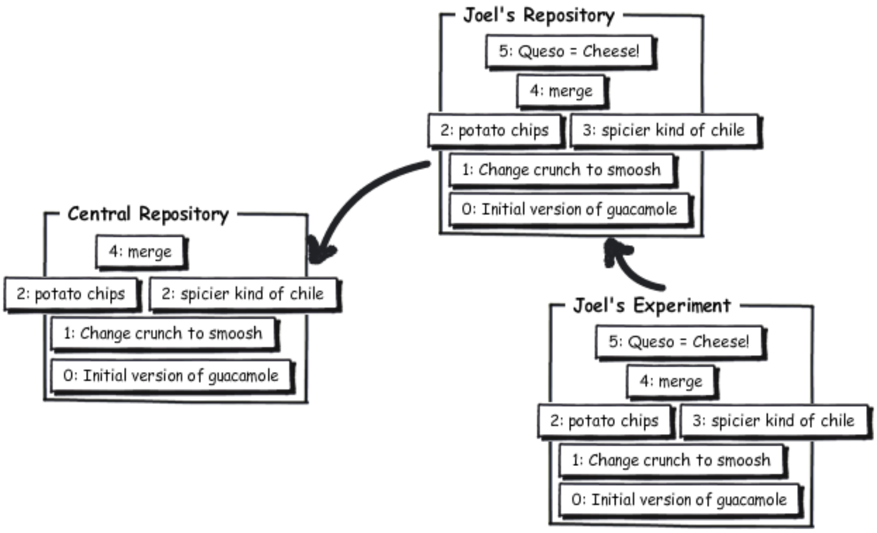
If I want to see what’s in changeset 5, I have to use the **hg update** command:



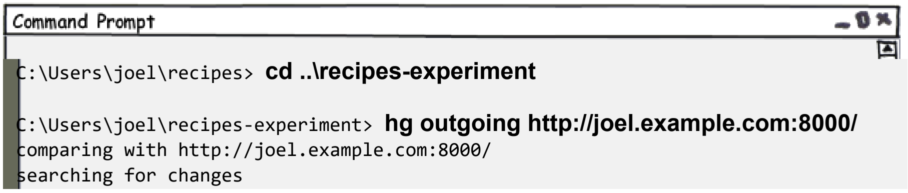
```
C:\Users\joel\recipes> hg up
1 files updated, 0 files merged, 0 files removed, 0 files unresolved

C:\Users\joel\recipes> hg parent
changeset:   5:9545248f3fc9
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 12:59:11 2010 -0500
summary:     Queso = Cheese!


C:\Users\joel\recipes> type guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with potato chips.

This recipe is really good served with QUESO.

QUESO is Spanish for "cheese," but in Texas,
it's just Kraft Slices melted in the microwave
with some salsa from a jar. MMM!
```

See what happened here? The changes came in, but they were on top of the version I was working on. **push** and **pull** just send changes from one repo to another—they don't affect the files I'm working on at the moment.

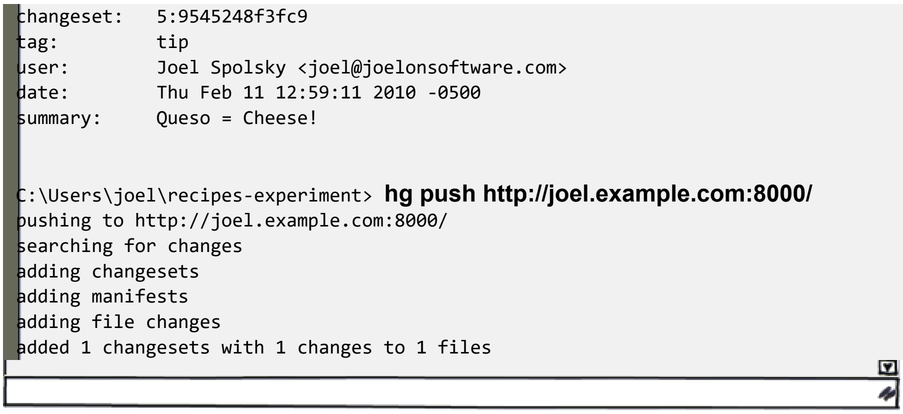
Right now here's the state of repositories:



Mercurial is flexible about moving changes around from repository to repository. You can push straight from the experimental repository into the central repository:

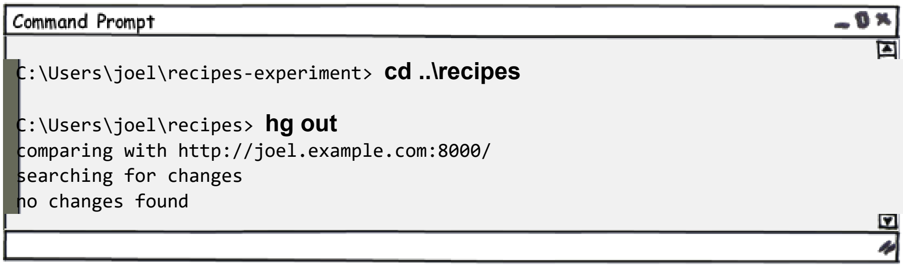


```
C:\Users\joel\recipes> cd ..\recipes-experiment

C:\Users\joel\recipes-experiment> hg outgoing http://joel.example.com:8000/
comparing with http://joel.example.com:8000/
searching for changes
```



```
changeset:   5:9545248f3fc9
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 12:59:11 2010 -0500
summary:     Queso = Cheese!


C:\Users\joel\recipes-experiment> hg push http://joel.example.com:8000/
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

That pushed change 5 from the experimental repo directly into the central repository. Now, if I go back to my repository, there's nothing left to push!

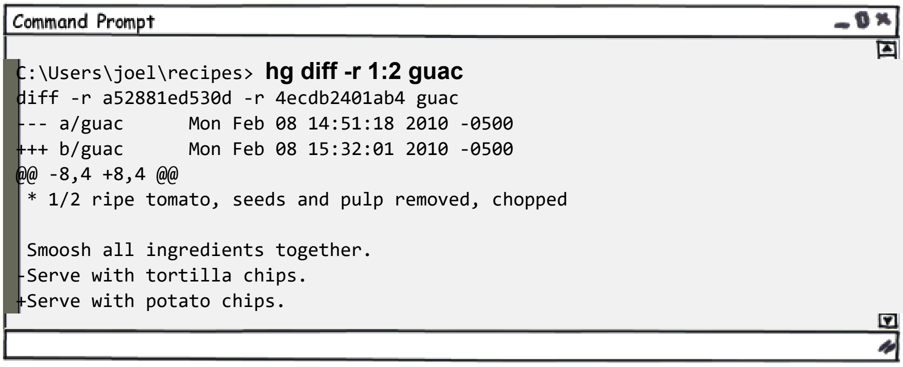


```
C:\Users\joel\recipes-experiment> cd ..\recipes

C:\Users\joel\recipes> hg out
comparing with http://joel.example.com:8000/
searching for changes
no changes found
```

That's because Mercurial knows that the central repo already got this particular changeset from somewhere else. That's really useful, because otherwise it would try to apply it again, and it would be massively confused.

After they made a job offer to designer Jim, he said he would start work right away, but then he didn't show up for two months. People had mostly forgotten about him and about the job offer, and when he showed up at the office for the first time to start work, looking rather sunburned, to be honest, nobody quite knew who he was or what was going on. It was pretty funny. He is kind of a generic looking guy. Eventually they figured it out, but since he was new, nobody had the guts to ask him what the hell had happened. Just like they never ask him about the bruises and scratches on his face. Whatever. We hate that guy.

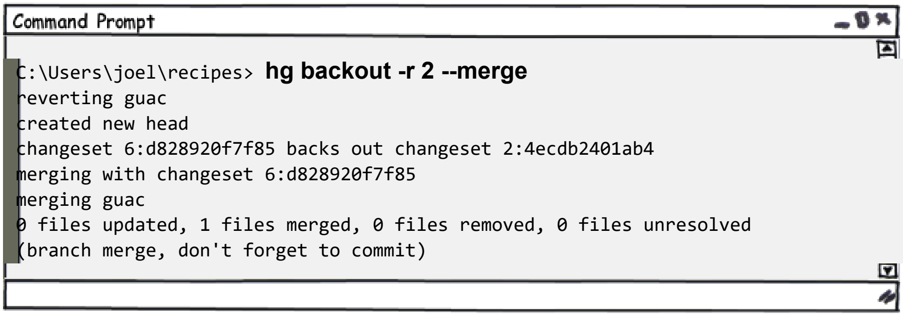
Sometimes it may happen that you discover that, months earlier, you made a mistake.



```
C:\Users\joel\recipes> hg diff -r 1:2 guac
diff -r a52881ed530d -r 4ecdb2401ab4 guac
--- a/guac      Mon Feb 08 14:51:18 2010 -0500
+++ b/guac      Mon Feb 08 15:32:01 2010 -0500
@@ -8,4 +8,4 @@
 * 1/2 ripe tomato, seeds and pulp removed, chopped

 Smoosh all ingredients together.
-Serve with tortilla chips.
+Serve with potato chips.
```

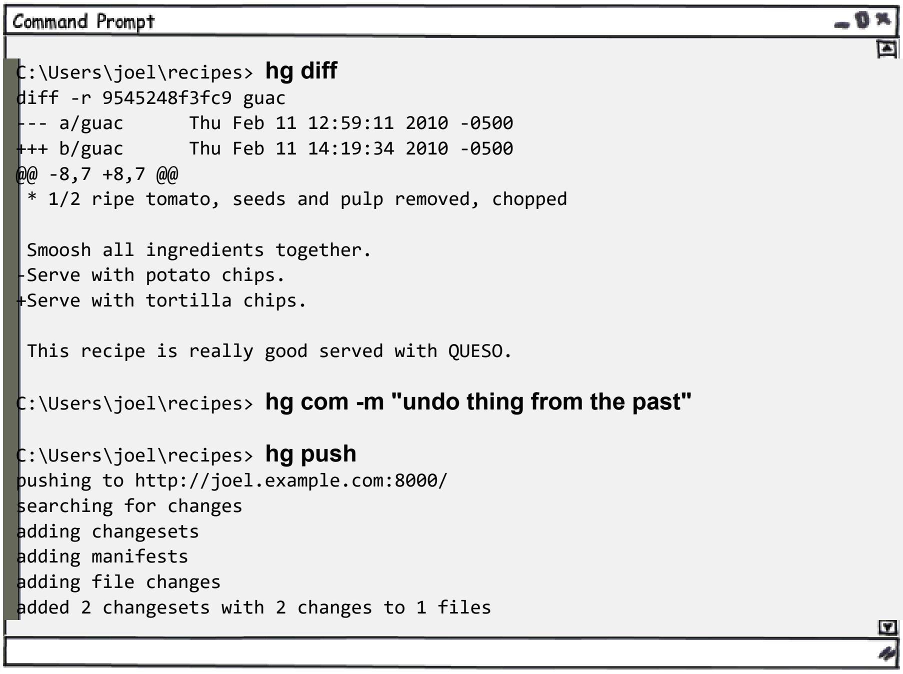
Potato chips? WTF?!

Mercurial can backout an old changeset from the past for you. It looks at the changeset, figures out the *opposite*, and does that to your current working directory. Let’s try backing out that old revision 2.



```
C:\Users\joel\recipes> hg backout -r 2 --merge
reverting guac
created new head
changeset 6:d828920f7f85 backs out changeset 2:4ecdb2401ab4
merging with changeset 6:d828920f7f85
merging guac
0 files updated, 1 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)
```

Holy crap, what just happened?



```
C:\Users\joel\recipes> hg diff
diff -r 9545248f3fc9 guac
--- a/guac      Thu Feb 11 12:59:11 2010 -0500
+++ b/guac      Thu Feb 11 14:19:34 2010 -0500
@@ -8,7 +8,7 @@
 * 1/2 ripe tomato, seeds and pulp removed, chopped

 Smoosh all ingredients together.
-Serve with potato chips.
+Serve with tortilla chips.

 This recipe is really good served with QUESO.

C:\Users\joel\recipes> hg com -m "undo thing from the past"

C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
```

Now, a lot of time may have passed. The chips might already be gone from the recipe. All kinds of spooky stuff might have happened that makes it impossible to merge in this change. In that case, you’re going to get merge conflicts, which you’re going to have to resolve somehow. We’ll talk about that in the next tutorial.

Test yourself

Here are the things you should know how to do after reading this tutorial:

1. Revert accidental changes, before and after checking in
2. Clone a repository locally for experiments
3. Push between repositories
4. Revert an old mistake that’s been in the repository for ages

« Home

Next, we talk about how to merge in Mercurial

Next »

Any questions?

If you have any questions about the material in this tutorial, no matter how newbie, ask them at the Kiln Knowledge Exchange.

About the author.

Joel Spolsky is the founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they're delighted. Fog Creek makes FogBugz, Kiln, and Fog Creek Copilot. Joel's blog Joel on Software is read by programmers everywhere.

Home Mercurial Joel on Software

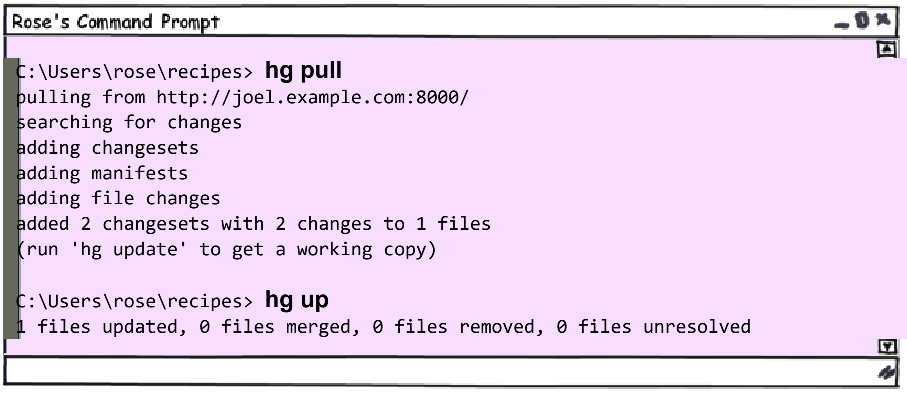
80
Hg
Init
200.59

Sometimes merges cause conflicts. Usually they’re easy to fix, but you do need to resolve them or you’ll have multiple heads. And who wants multiple heads?

Merging

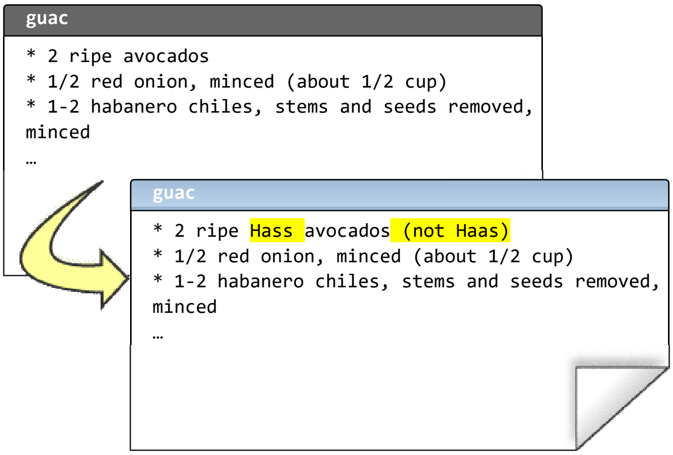
An important part of version control is coordinating the work of multiple people on the same code base.

Imagine that Rose and I both want to make a change to the guacamole recipe. Rose is improving the avocado quality. Before she starts, she’s going to pull all the latest changes from the central repository so that she’s totally up to date.


Rose's Command Prompt

```
C:\Users\rose\recipes> hg pull
pulling from http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
(run 'hg update' to get a working copy)

C:\Users\rose\recipes> hg up
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Now an edit:


guac

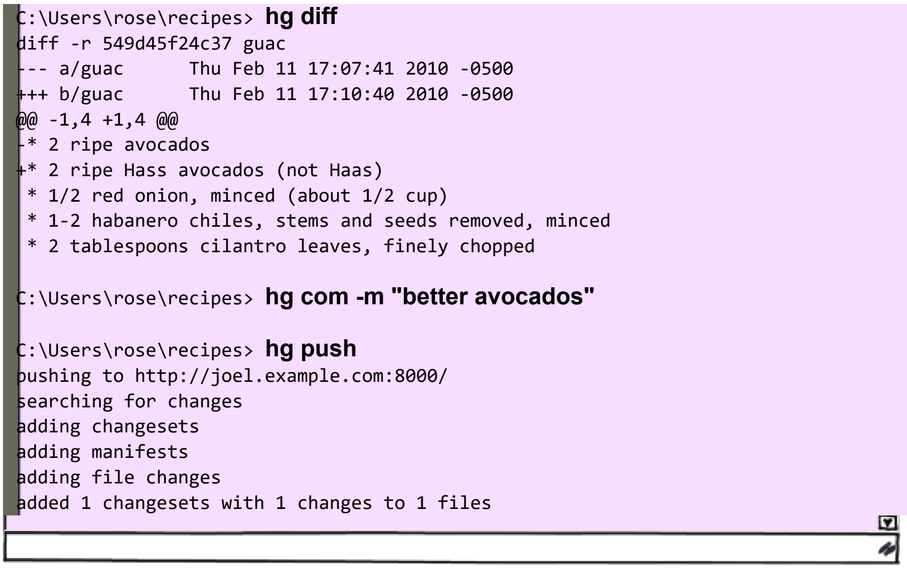
```
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed,
minced
…
```

guac

```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed,
minced
…
```

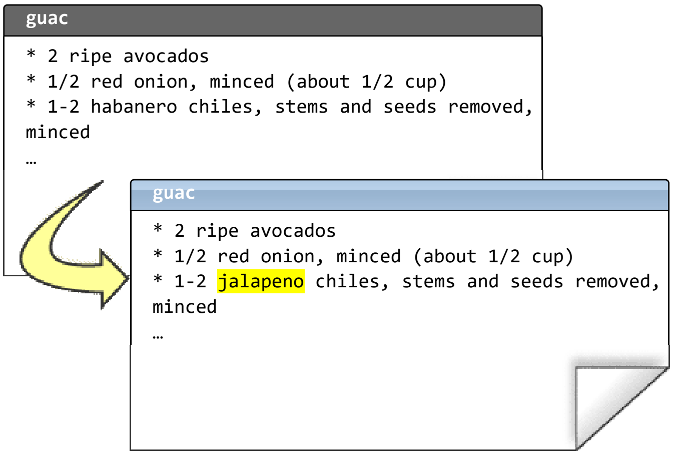
She commits and pushes the change off to the central repository:

Rose's Command Prompt

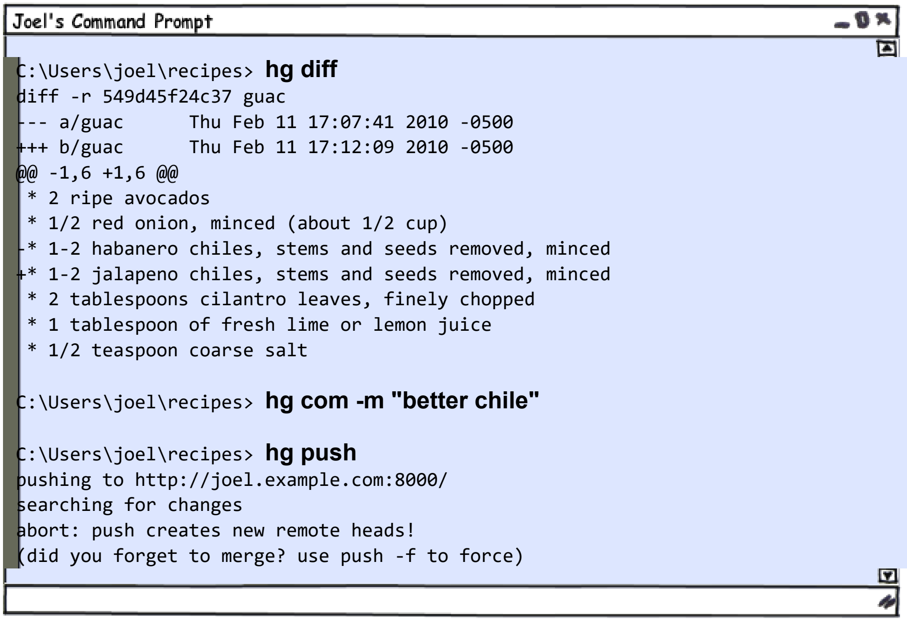


```
C:\Users\rose\recipes> hg diff
diff -r 549d45f24c37 guac
--- a/guac      Thu Feb 11 17:07:41 2010 -0500
+++ b/guac      Thu Feb 11 17:10:40 2010 -0500
@@ -1,4 +1,4 @@
-* 2 ripe avocados
+* 2 ripe Hass avocados (not Haas)
 * 1/2 red onion, minced (about 1/2 cup)
 * 1-2 habanero chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped

C:\Users\rose\recipes> hg com -m "better avocados"

C:\Users\rose\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

Simultaneously, I make a change in a different part of the file:

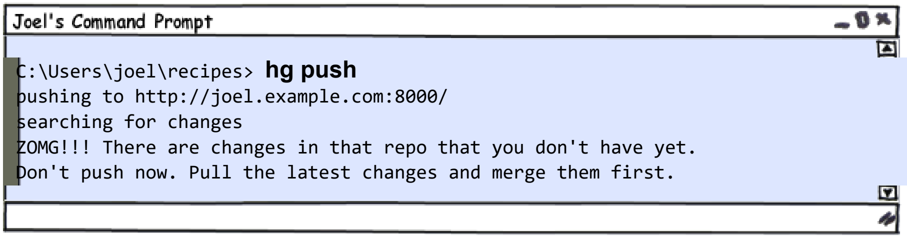

guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 habanero chiles, stems and seeds removed, minced
…
guac
* 2 ripe avocados
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed, minced
…

I can commit, but I can’t push to the central repository.



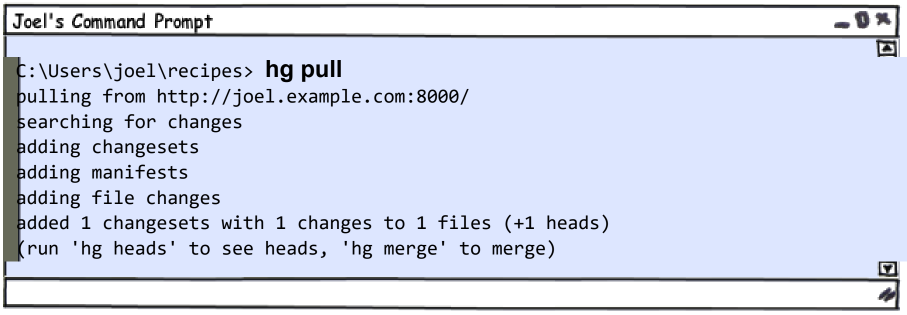
```
C:\Users\joel\recipes> hg diff
diff -r 549d45f24c37 guac
--- a/guac      Thu Feb 11 17:07:41 2010 -0500
+++ b/guac      Thu Feb 11 17:12:09 2010 -0500
@@ -1,6 +1,6 @@
 * 2 ripe avocados
 * 1/2 red onion, minced (about 1/2 cup)
-* 1-2 habanero chiles, stems and seeds removed, minced
+* 1-2 jalapeno chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped
 * 1 tablespoon of fresh lime or lemon juice
 * 1/2 teaspoon coarse salt

C:\Users\joel\recipes> hg com -m "better chile"

C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
abort: push creates new remote heads!
(did you forget to merge? use push -f to force)
```

This is probably the least useful error message in Mercurial. What it should say is:



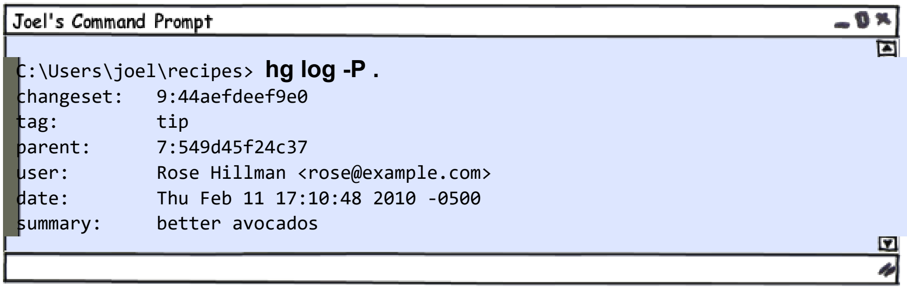
```
C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
ZOMG!!! There are changes in that repo that you don't have yet.
Don't push now. Pull the latest changes and merge them first.
```

And indeed that is what I will do:



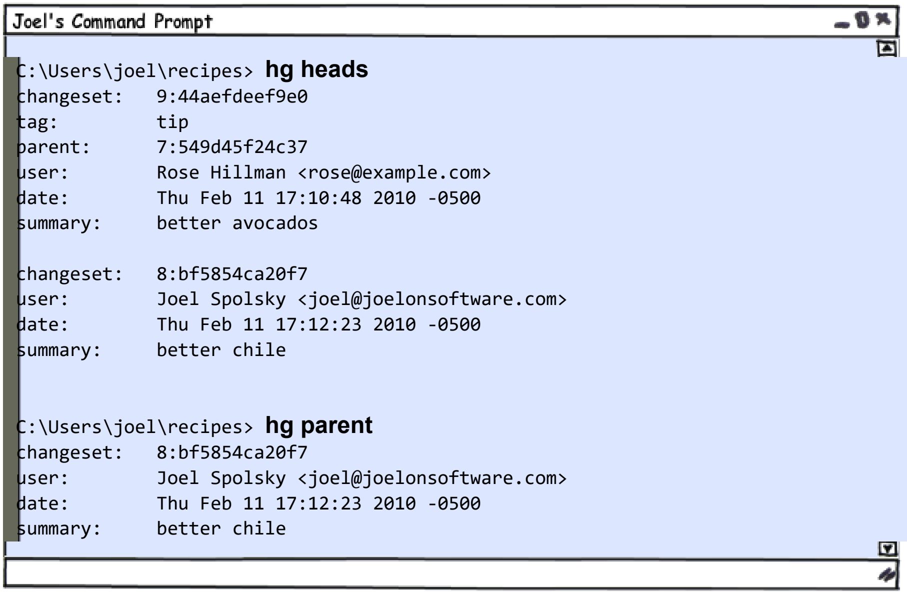
```
C:\Users\joel\recipes> hg pull
pulling from http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)
```

Wondering what just arrived? The **hg log -P .** command is a handy way to find out.



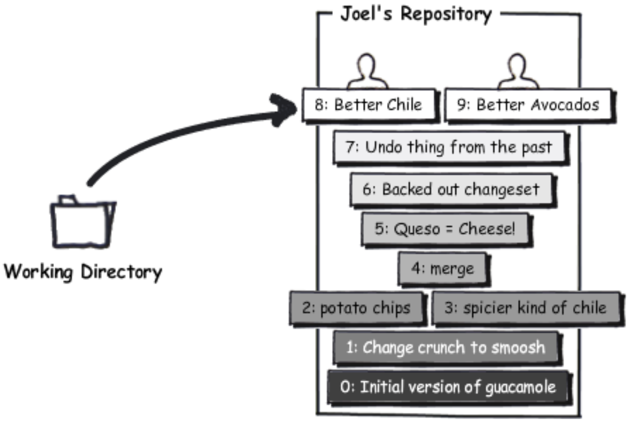
```
C:\Users\joel\recipes> hg log -P .
changeset:   9:44aefdeef9e0
tag:         tip
parent:      7:549d45f24c37
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 17:10:48 2010 -0500
summary:     better avocados
```

Indeed, it's Rose's change from earlier. What's going on with my repository right now?

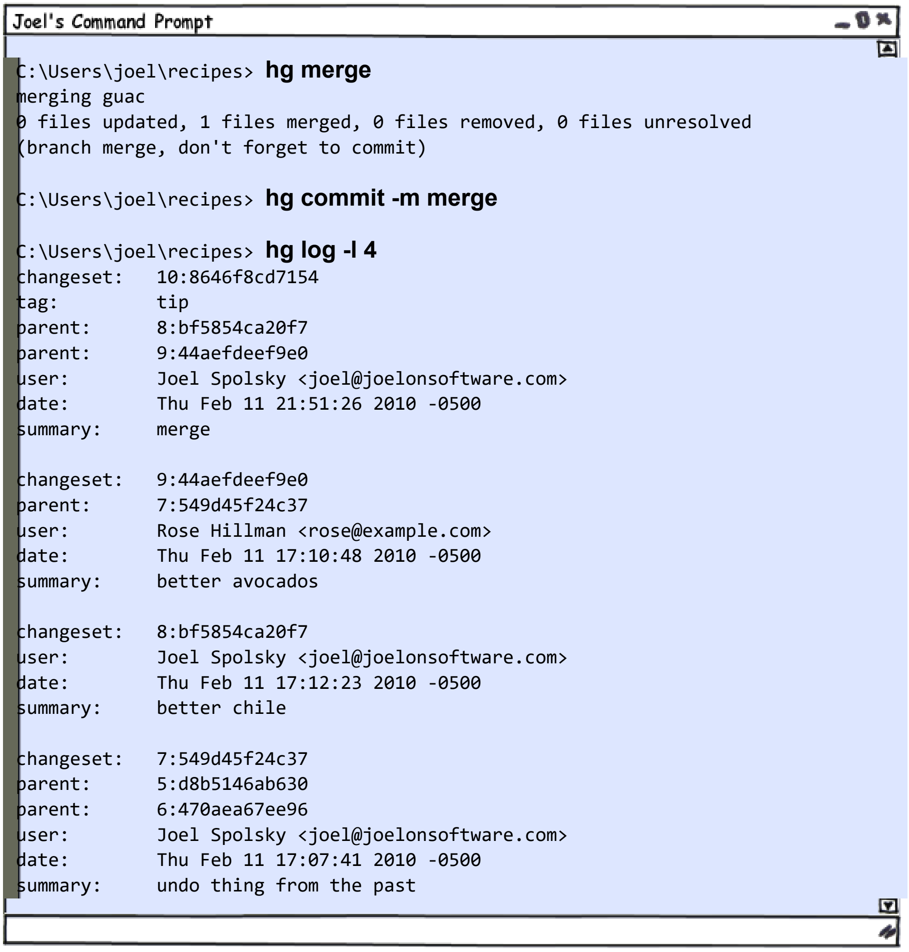


```
C:\Users\joel\recipes> hg heads
changeset:   9:44aefdeef9e0
tag:         tip
parent:      7:549d45f24c37
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 17:10:48 2010 -0500
summary:     better avocados

changeset:   8:bf5854ca20f7
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 17:12:23 2010 -0500
summary:     better chile


C:\Users\joel\recipes> hg parent
changeset:   8:bf5854ca20f7
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 17:12:23 2010 -0500
summary:     better chile
```

I have “multiple heads.” Essentially, my repository looks like this:

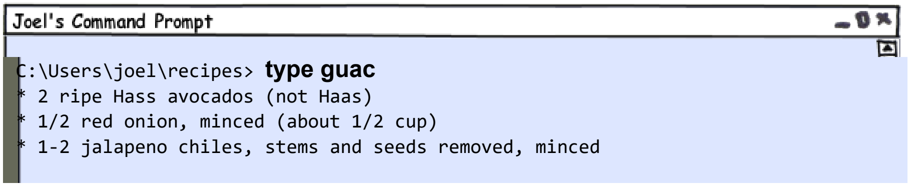


See the two heads? They came about because when Rose made her change, she was working off of changeset 7, and when I made my change, I was working off of changeset 7, too. So now a merge is needed. [Ed: Never use passive voice!] *I need to merge.*

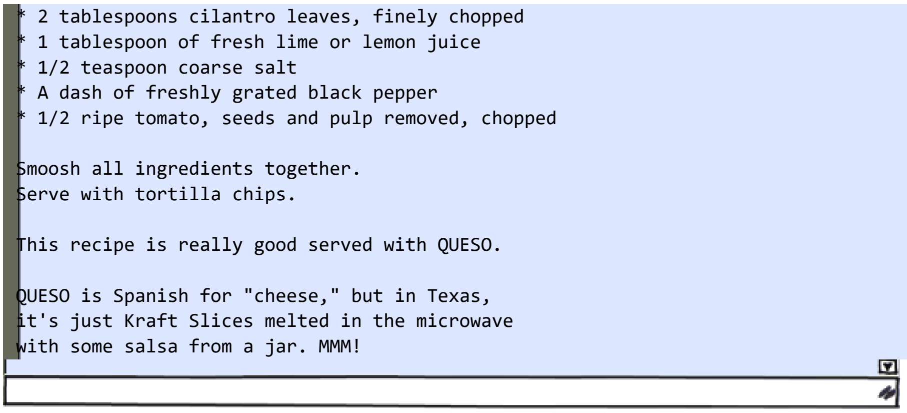


```
C:\Users\joel\recipes> hg merge
merging guac
0 files updated, 1 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\joel\recipes> hg commit -m merge

C:\Users\joel\recipes> hg log -l 4
changeset:   10:8646f8cd7154
tag:         tip
parent:      8:bf5854ca20f7
parent:      9:44aefdeef9e0
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 21:51:26 2010 -0500
summary:     merge

changeset:   9:44aefdeef9e0
parent:      7:549d45f24c37
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 17:10:48 2010 -0500
summary:     better avocados

changeset:   8:bf5854ca20f7
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 17:12:23 2010 -0500
summary:     better chile

changeset:   7:549d45f24c37
parent:      5:d8b5146ab630
parent:      6:470aea67ee96
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 17:07:41 2010 -0500
summary:     undo thing from the past
```

The merge command, **hg merge**, took the two heads and combined them. Then it left the result in my working directory. It did not commit it. That gives me a chance to check that the merge is correct:

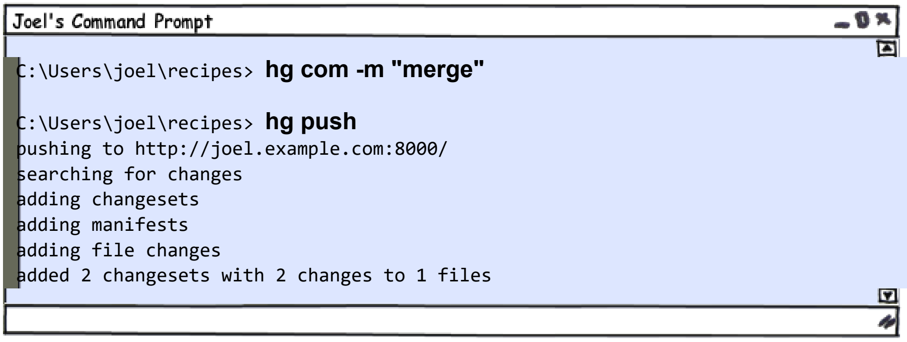


```
C:\Users\joel\recipes> type guac
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed, minced
```



```
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped

Smoosh all ingredients together.
Serve with tortilla chips.

This recipe is really good served with QUESO.

QUESO is Spanish for "cheese," but in Texas,
it's just Kraft Slices melted in the microwave
with some salsa from a jar. MMM!
```

That looks right; the avocados are Hass and the chiles are Jalapenos. So I’ll go ahead and commit and push that to the server.



```
C:\Users\joel\recipes> hg com -m "merge"

C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
```

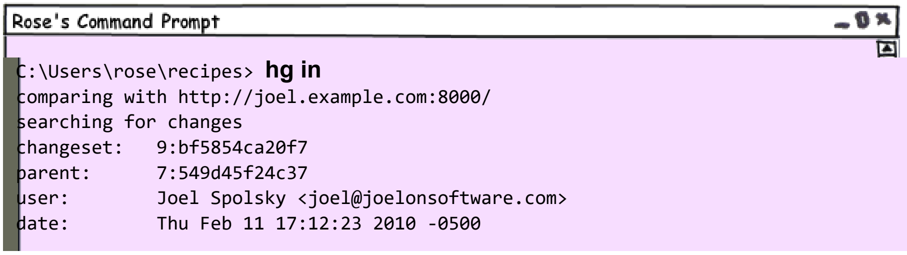
I’m pushing two changesets: my original jalapeno change, and then the merge, which is its own changeset.

Notice that nothing about our two changes conflicted, since Rose and I were working on different parts of the recipe. So the merge was super duper easy. That’s the most common case, because in most organizations, each programmer is assigned to work on a different part of the code.

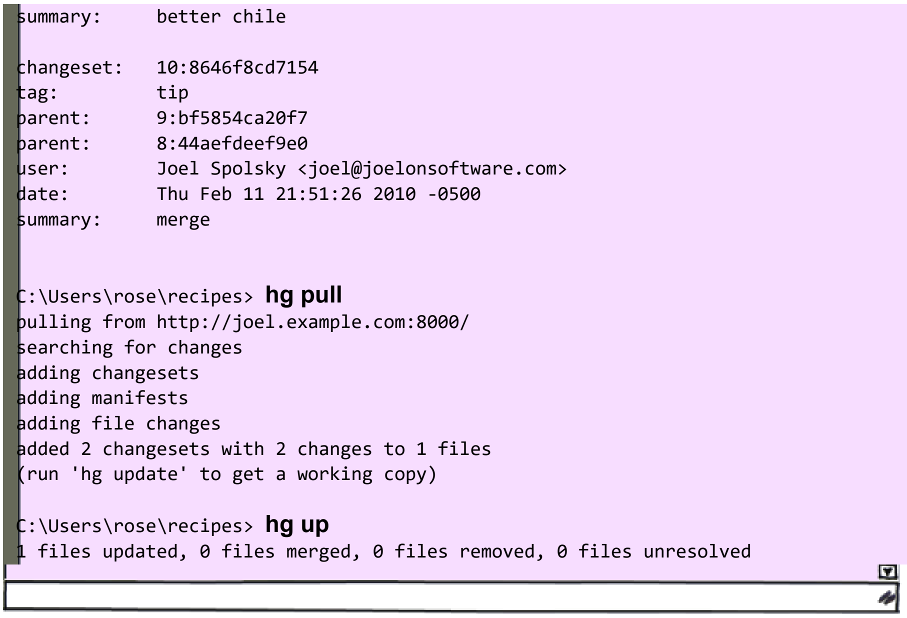
Sometimes you have a dysfunctional organization where nobody is willing to really put their foot down about who is supposed to work on what. This can cause sudden and often unexplained feelings of sadness among the programming staff. This is hard to detect. Symptoms can include: programmers locking themselves in bathrooms, programmers locking themselves in the server closet, high turnover, muted sobbing sounds in the cube farm, and sudden eardrum trauma caused by the sound of repeated firing of a military-class assault rifle.

BUT, even in the best-managed and healthiest organizations, merge conflicts do sometimes occur, and Mercurial will require the merging person to resolve the conflict. Let’s see what that looks like.

First… I want to bring Rose up to speed with my jalapeno changes:



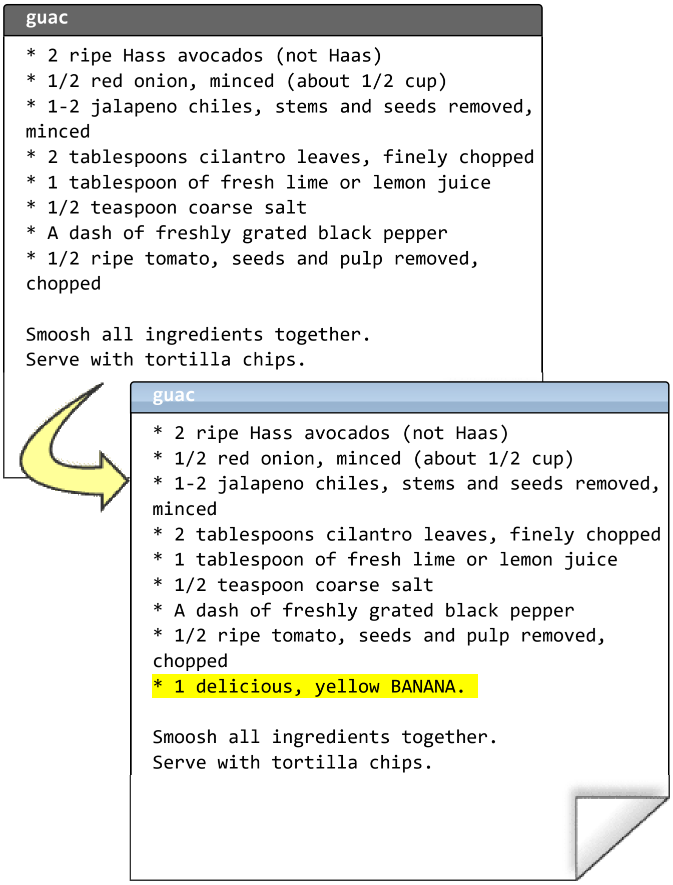
```
C:\Users\rose\recipes> hg in
comparing with http://joel.example.com:8000/
searching for changes
changeset:   9:bf5854ca20f7
parent:      7:549d45f24c37
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 17:12:23 2010 -0500
```



```
summary:     better chile

changeset:   10:8646f8cd7154
tag:         tip
parent:      9:bf5854ca20f7
parent:      8:44aefdeef9e0
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 21:51:26 2010 -0500
summary:     merge


C:\Users\rose\recipes> hg pull
pulling from http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
(run 'hg update' to get a working copy)

C:\Users\rose\recipes> hg up
1 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Now we’re going to see what happens when you have a gen-yoo-ine conflict: we’re both going to muck with the ingredients a bit.

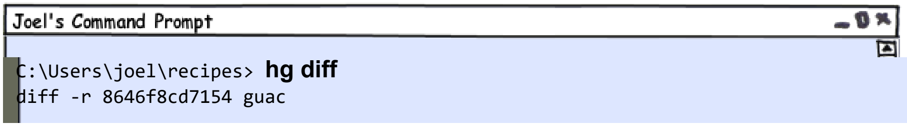
I added a banana:



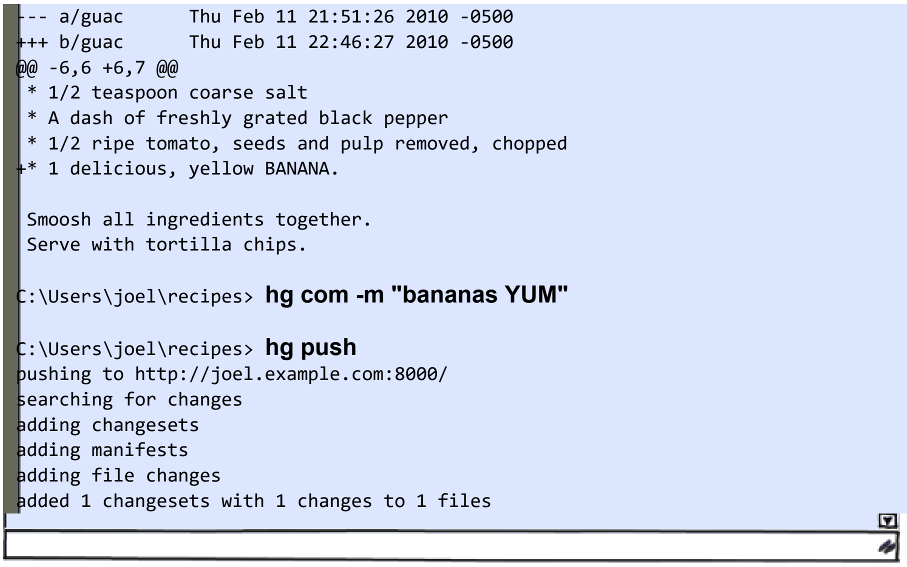
```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Smoosh all ingredients together.
Serve with tortilla chips.
```

```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped
* 1 delicious, yellow BANANA.

Smoosh all ingredients together.
Serve with tortilla chips.
```

I checked in the banana change first:

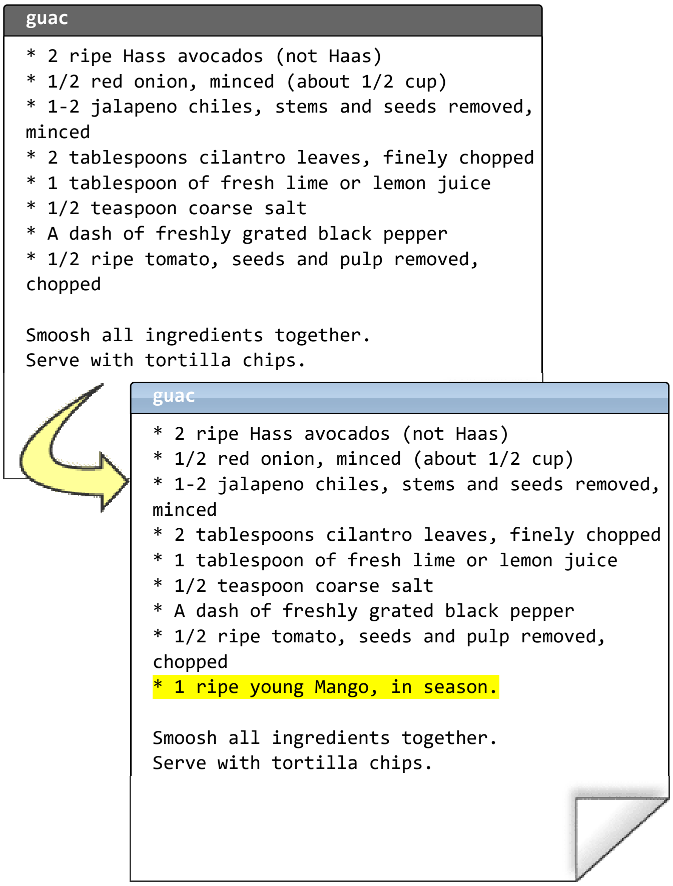


```
C:\Users\joel\recipes> hg diff
diff -r 8646f8cd7154 guac
```



```
--- a/guac      Thu Feb 11 21:51:26 2010 -0500
+++ b/guac      Thu Feb 11 22:46:27 2010 -0500
@@ -6,6 +6,7 @@
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
+* 1 delicious, yellow BANANA.

 Smoosh all ingredients together.
 Serve with tortilla chips.

C:\Users\joel\recipes> hg com -m "bananas YUM"

C:\Users\joel\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files
```

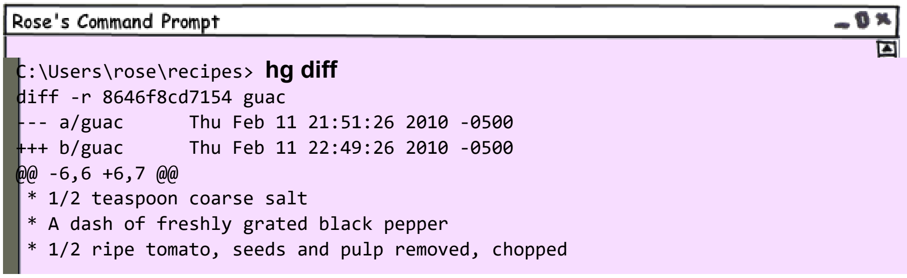
And Rose, bless her heart, added a MANGO on the EXACT SAME LINE.



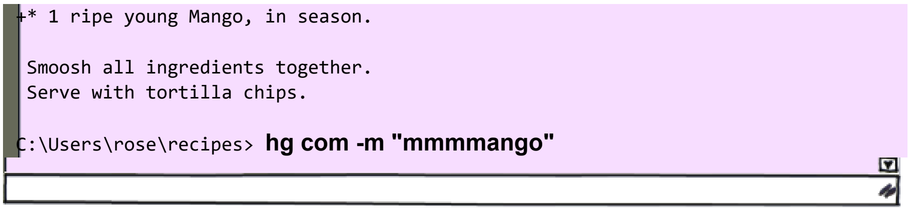
```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped

Smoosh all ingredients together.
Serve with tortilla chips.
```

```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped
* 1 ripe young Mango, in season.

Smoosh all ingredients together.
Serve with tortilla chips.
```

“Ripe young” mango, indeed.

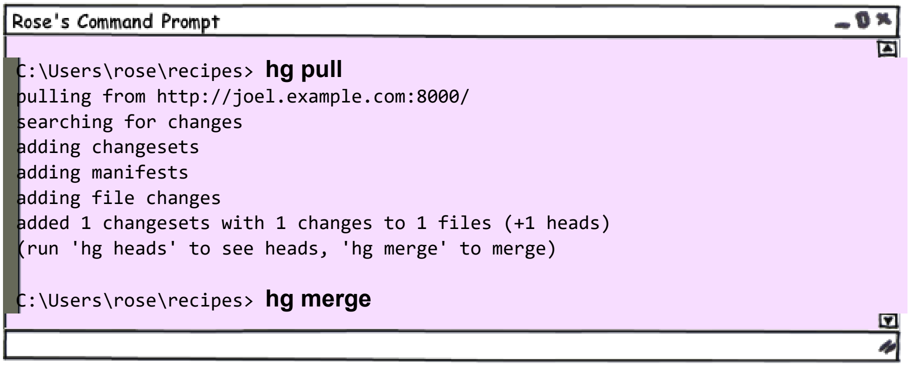


```
C:\Users\rose\recipes> hg diff
diff -r 8646f8cd7154 guac
--- a/guac      Thu Feb 11 21:51:26 2010 -0500
+++ b/guac      Thu Feb 11 22:49:26 2010 -0500
@@ -6,6 +6,7 @@
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
```



```
+* 1 ripe young Mango, in season.

 Smoosh all ingredients together.
 Serve with tortilla chips.

C:\Users\rose\recipes> hg com -m "mmmmango"
```

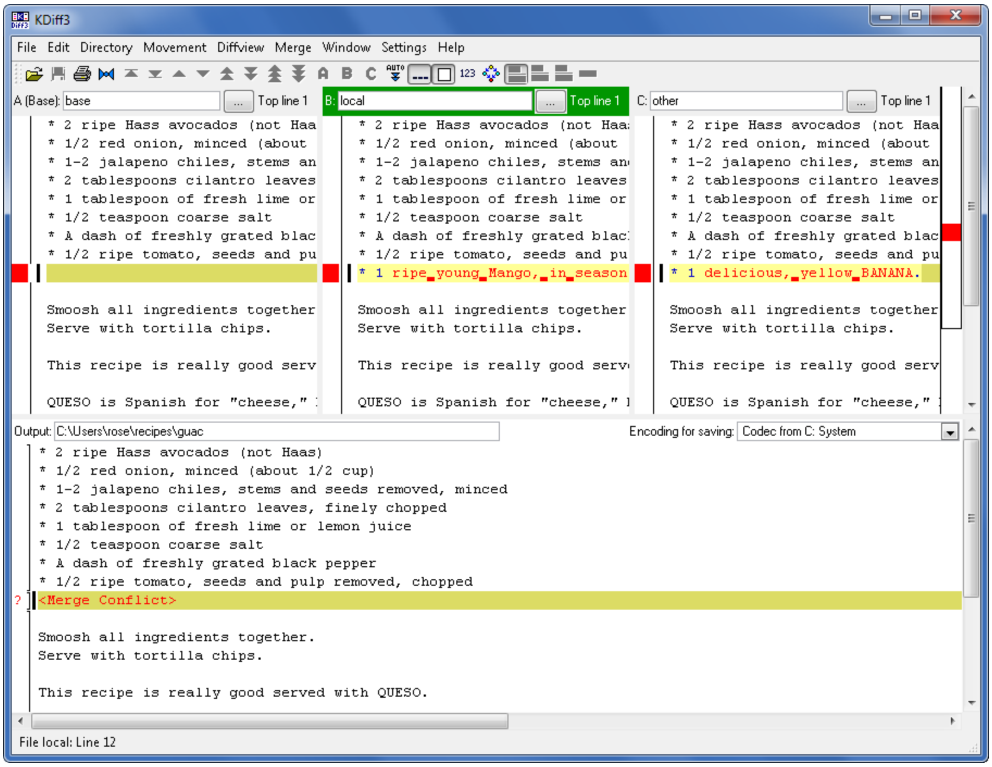
This time I got my change in first so Rose is going to have to merge. HA!



Rose's Command Prompt

```
C:\Users\rose\recipes> hg pull
pulling from http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 1 changesets with 1 changes to 1 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)

C:\Users\rose\recipes> hg merge
```

And suddenly, the conflict is detected, and up pops some kind of a graphical merge-conflict-resolution tool, with a user interface that only its mother could love, but they are usually quite good at what they do once you figure it out. A common merge conflict resolution tool these days is KDiff3, which shows Rose the following:



KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
...
Top line 1
...
Top line 1
...
Top line 1

A (Base): base

```
 * 2 ripe Hass avocados (not Haa
 * 1/2 red onion, minced (about
 * 1-2 jalapeno chiles, stems an
 * 2 tablespoons cilantro leaves
 * 1 tablespoon of fresh lime or
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated blac
 * 1/2 ripe tomato, seeds and pu


 Smoosh all ingredients together
 Serve with tortilla chips.

 This recipe is really good serv

 QUESO is Spanish for "cheese," 
```

B: local

```
 * 2 ripe Hass avocados (not Haa
 * 1/2 red onion, minced (about
 * 1-2 jalapeno chiles, stems an
 * 2 tablespoons cilantro leaves
 * 1 tablespoon of fresh lime or
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated blac
 * 1/2 ripe tomato, seeds and pu
 * 1 ripe_young_Mango,_in_season

 Smoosh all ingredients together
 Serve with tortilla chips.

 This recipe is really good serv

 QUESO is Spanish for "cheese," I
```

C: other

```
 * 2 ripe Hass avocados (not Haa
 * 1/2 red onion, minced (about
 * 1-2 jalapeno chiles, stems an
 * 2 tablespoons cilantro leaves
 * 1 tablespoon of fresh lime or
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated blac
 * 1/2 ripe tomato, seeds and pu
 * 1 delicious,_yellow_BANANA.

 Smoosh all ingredients together
 Serve with tortilla chips.

 This recipe is really good serv

 QUESO is Spanish for "cheese,"
```

Output: C:\Users\rose\recipes\guac

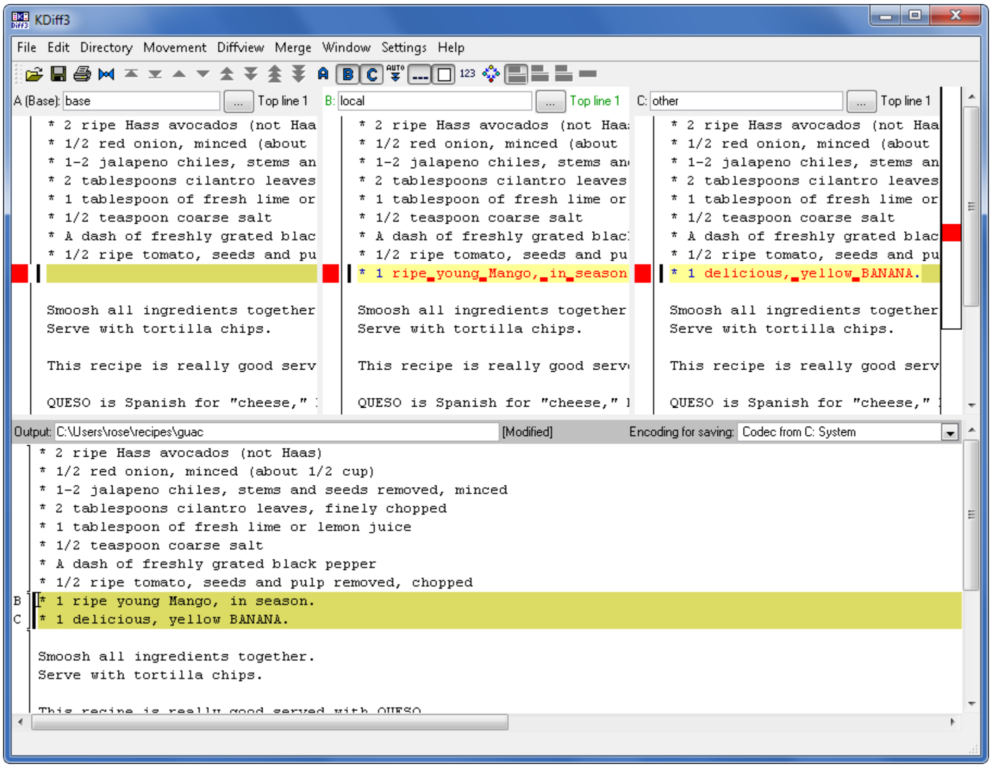
Encoding for saving: Codec from C: System

```
 * 2 ripe Hass avocados (not Haas)
 * 1/2 red onion, minced (about 1/2 cup)
 * 1-2 jalapeno chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped
 * 1 tablespoon of fresh lime or lemon juice
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
?<Merge Conflict>

 Smoosh all ingredients together.
 Serve with tortilla chips.

 This recipe is really good served with QUESO.
```

File local: Line 12

In KDiff3, you see four panes. The top left is the original file. Top center shows Rose her version. Top right shows Rose my version. The bottom pane is an editor where Rose constructs a merged file with the conflicts resolved.

Fixing conflicts is a relatively simple matter of going through each conflict and choosing how to resolve it. Rose goes crazy and decides that Banana Mango Guacamole couldn't be that bad:


KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A (Base): base
Top line 1
B: local
Top line 1
C: other
Top line 1
Output: C:\Users\rose\recipes\guac
[Modified]
Encoding for saving: Codec from C: System

```
 * 2 ripe Hass avocados (not Haa
 * 1/2 red onion, minced (about
 * 1-2 jalapeno chiles, stems an
 * 2 tablespoons cilantro leaves
 * 1 tablespoon of fresh lime or
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated blac
 * 1/2 ripe tomato, seeds and pu

 Smoosh all ingredients together
 Serve with tortilla chips.

 This recipe is really good serv

 QUESO is Spanish for "cheese,"
```

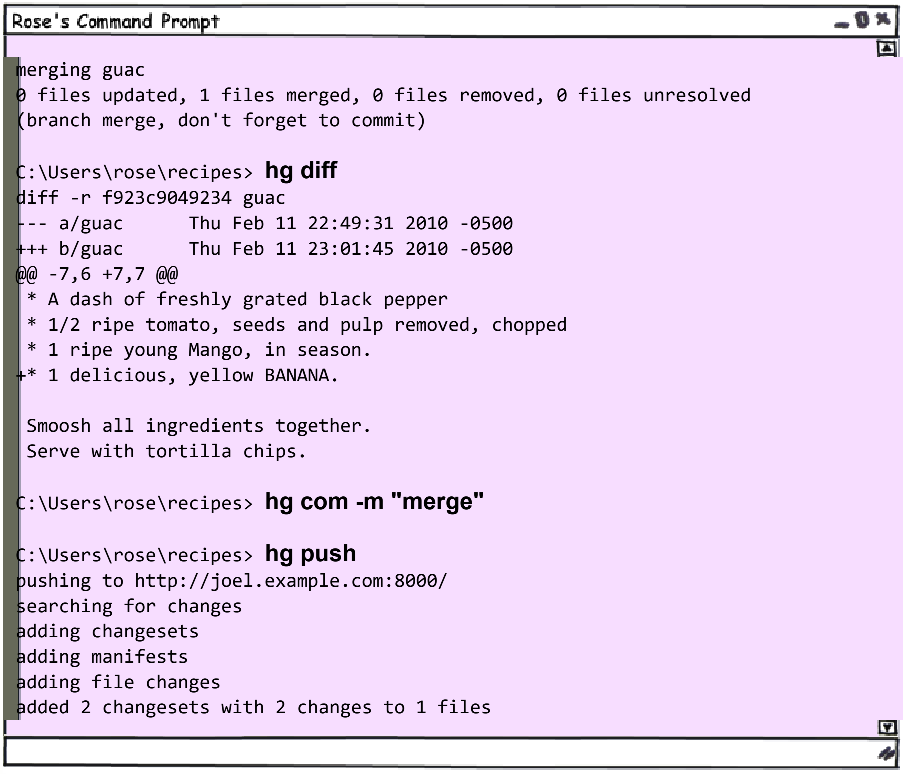
```
 * 2 ripe Hass avocados (not Haa
 * 1/2 red onion, minced (about
 * 1-2 jalapeno chiles, stems an
 * 2 tablespoons cilantro leaves
 * 1 tablespoon of fresh lime or
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated blac
 * 1/2 ripe tomato, seeds and pu
 * 1 ripe young Mango, in season

 Smoosh all ingredients together
 Serve with tortilla chips.

 This recipe is really good serv

 QUESO is Spanish for "cheese,"
```

```
 * 2 ripe Hass avocados (not Haa
 * 1/2 red onion, minced (about
 * 1-2 jalapeno chiles, stems an
 * 2 tablespoons cilantro leaves
 * 1 tablespoon of fresh lime or
 * 1/2 teaspoon coarse salt
 * A dash of freshly grated blac
 * 1/2 ripe tomato, seeds and pu
 * 1 delicious, yellow BANANA.

 Smoosh all ingredients together
 Serve with tortilla chips.

 This recipe is really good serv

 QUESO is Spanish for "cheese,"
```

```
  * 2 ripe Hass avocados (not Haas)
  * 1/2 red onion, minced (about 1/2 cup)
  * 1-2 jalapeno chiles, stems and seeds removed, minced
  * 2 tablespoons cilantro leaves, finely chopped
  * 1 tablespoon of fresh lime or lemon juice
  * 1/2 teaspoon coarse salt
  * A dash of freshly grated black pepper
  * 1/2 ripe tomato, seeds and pulp removed, chopped
B * 1 ripe young Mango, in season.
C * 1 delicious, yellow BANANA.

  Smoosh all ingredients together.
  Serve with tortilla chips.

  This recipe is really good served with QUESO
```

By the way, did I tell you that Rose seems to be dating? The other day she was seen leaving work with a guy that sort-of looked like Dennis Franz. She's been in the best mood anyone has seen her in years.

Rose saves her changes and exits KDiff3.



```
merging guac
0 files updated, 1 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\rose\recipes> hg diff
diff -r f923c9049234 guac
--- a/guac      Thu Feb 11 22:49:31 2010 -0500
+++ b/guac      Thu Feb 11 23:01:45 2010 -0500
@@ -7,6 +7,7 @@
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
 * 1 ripe young Mango, in season.
+* 1 delicious, yellow BANANA.

 Smoosh all ingredients together.
 Serve with tortilla chips.

C:\Users\rose\recipes> hg com -m "merge"

C:\Users\rose\recipes> hg push
pushing to http://joel.example.com:8000/
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 1 files
```

And the conflict is resolved.

One thing you should keep in mind: you do not have to merge on anybody's schedule. You can **hg pull** whenever you want, and if you don't feel like merging the conflicts, you're free to keep working, and committing, happily, until you have time to think about the merge.

Test yourself

Here are the things you should know how to do after reading this tutorial:

1. Work on a code base with other people
2. Get their changes
3. Push your changes
4. Resolve merge conflicts that may come up from time to time
5. Diagnose certain classes of programmer melancholy

« Home

Next, we talk about repository architecture

Next »

Any questions?

If you have any questions about the material in this tutorial, no matter how newbie, ask them at the Kiln Knowledge Exchange.

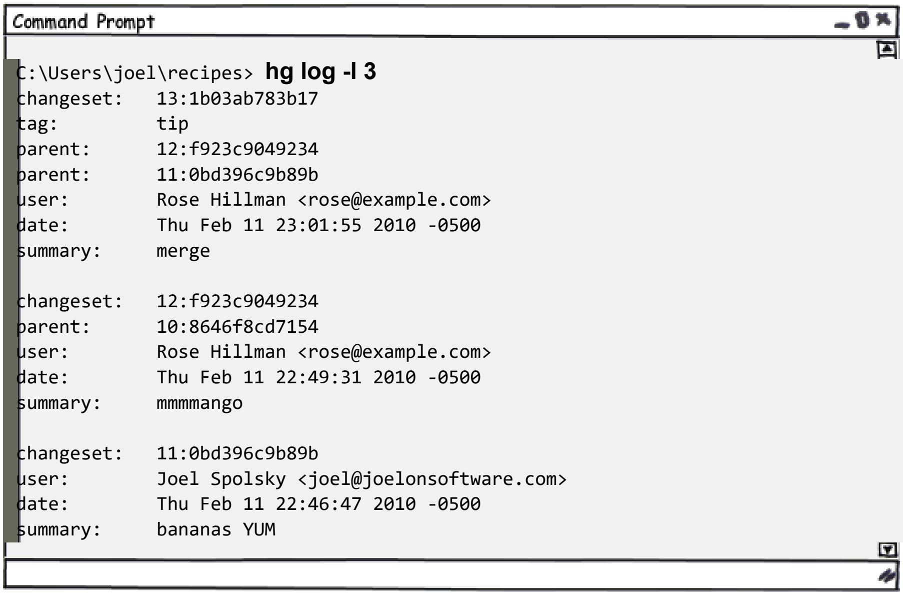
About the author.

Joel Spolsky is the founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they're delighted. Fog Creek makes FogBugz, Kiln, and Fog Creek Copilot. Joel's blog Joel on Software is read by programmers everywhere.

Mercurial gives you a great deal of flexibility in setting up repositories. Since merges work so well, you can depend on them, which means you can keep special-purpose repos around to match your development process.

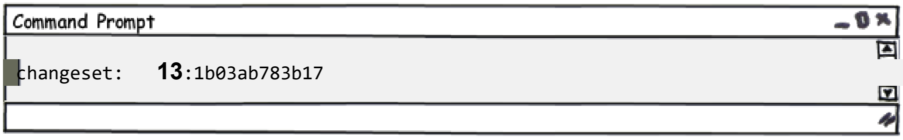
Repository Architecture

Our recipe is getting pretty good:



```
C:\Users\joel\recipes> hg log -l 3
changeset:   13:1b03ab783b17
tag:         tip
parent:      12:f923c9049234
parent:      11:0bd396c9b89b
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 23:01:55 2010 -0500
summary:     merge

changeset:   12:f923c9049234
parent:      10:8646f8cd7154
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 22:49:31 2010 -0500
summary:     mmmmango

changeset:   11:0bd396c9b89b
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Thu Feb 11 22:46:47 2010 -0500
summary:     bananas YUM
```

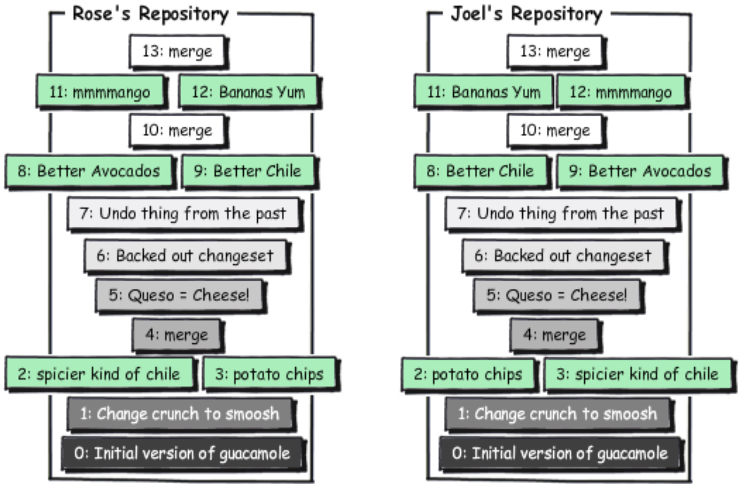
Let’s look more closely at that changeset number:



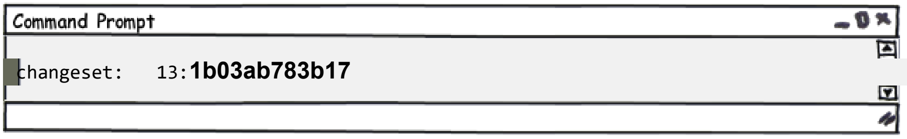
```
changeset:   13:1b03ab783b17
```

The first part of the number, 13, is short and convenient. There’s only one problem… it’s not reliable!

When people on the team work separately and then merge their work, those short numbers get out of sync:



So, for all intents and purposes, I can't tell people "OK, let's ship the revision based on changeset 13", because they might have a different idea of what 13 is. That's why there's that crazy hexadecimal number.

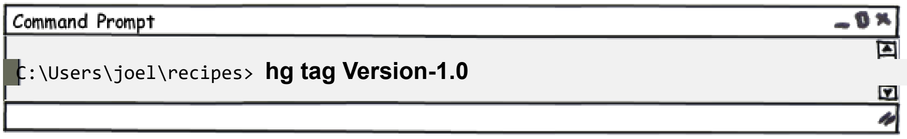


```
changeset:   13:1b03ab783b17
```

The hexadecimal number *is* consistent across all repositories and will never change.

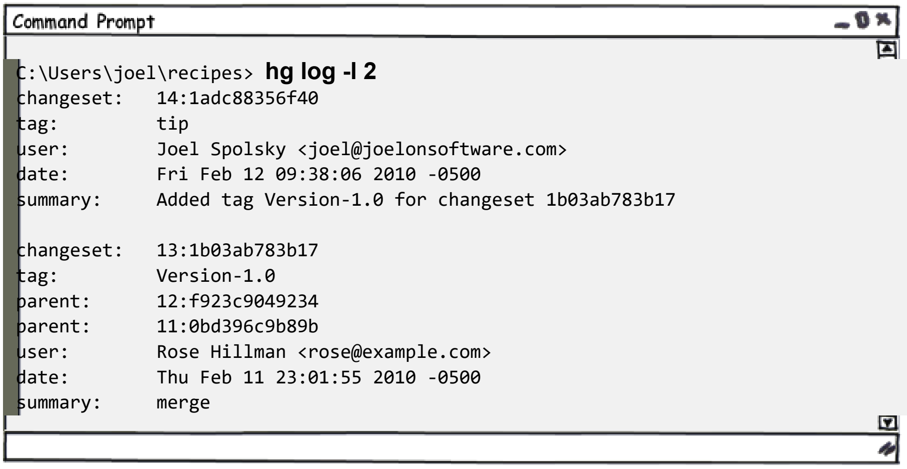
OK, so now I could tell people, "Hey, we're shipping today! Changeset number 1b03ab783b17! Wouldn't it be nice if I could give this changeset a *name*?

Well, you can. It's called a **tag**.



```
C:\Users\joel\recipes> hg tag Version-1.0
```

Let's look at the log now:

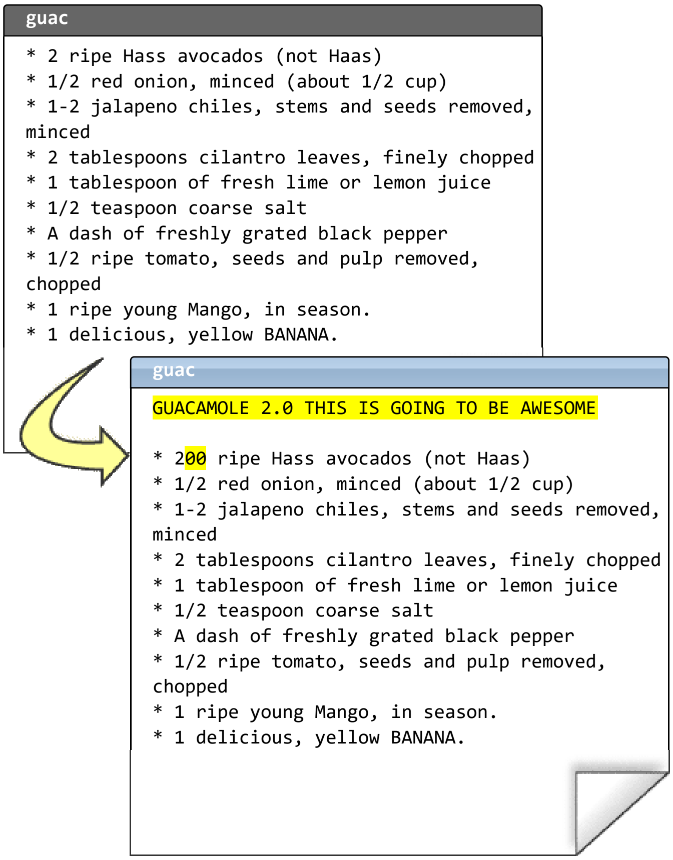


```
C:\Users\joel\recipes> hg log -l 2
changeset:   14:1adc88356f40
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 09:38:06 2010 -0500
summary:     Added tag Version-1.0 for changeset 1b03ab783b17

changeset:   13:1b03ab783b17
tag:         Version-1.0
parent:      12:f923c9049234
parent:      11:0bd396c9b89b
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 23:01:55 2010 -0500
summary:     merge
```

Notice that the the very act of adding the tag was a changeset, and it got committed automatically for me. So now, every time I want to refer to the version of the code that we shipped, I can use **Version-1.0** instead of **1b03ab783b17**.

The CEO came down from the 31st floor for the office ship party, with a crate of rather expensive looking sparkling wine. Stan got a little bit drunk. Well, not a little. Nobody had ever seen anything like it. He was taking off his shirt, showing off his muscles and not insignificant flab, and trying to impress some women from the marketing department. “I’m gonna do pullups from the lights,” he brags (we have these long fluorescent lights). So he jumps up, grabs the fixture, and of course, pulls down the whole thing immediately, since it was a ten pound light fixture hanging from a couple of thin pieces of piano wire, and his weight was somewhere in that hotly contested zone between 290 and 300 pounds. He pulls down the whole light and a lot of ceiling tiles, too, shattered glass and acoustic-tile-material everywhere, and lands smack on his tailbone whining about how he’s going to sue the company for making an unsafe work environment.

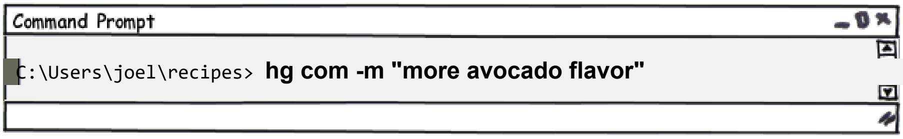
The rest of us went back to our cubes, to work on Guac 2.0.



```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped
* 1 ripe young Mango, in season.
* 1 delicious, yellow BANANA.
```

```
GUACAMOLE 2.0 THIS IS GOING TO BE AWESOME

* 200 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped
* 1 ripe young Mango, in season.
* 1 delicious, yellow BANANA.
```

Commit:

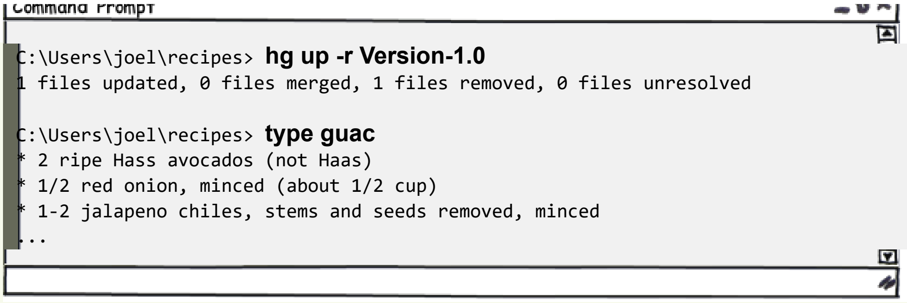


```
C:\Users\joel\recipes> hg com -m "more avocado flavor"
```

Needless to say, the recipe here is in a very unsettled state. It hasn’t been tested or anything. And then, the (one) customer calls up.

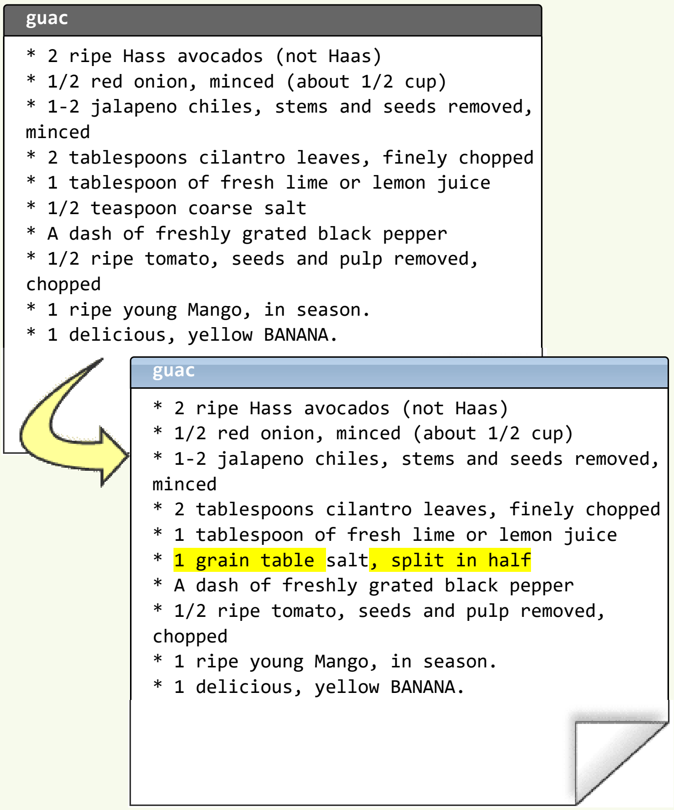
“It’s too salty!” he whimpers. And no, he doesn’t want to wait for version 2.0 for a fix.

Luckily, we’ve got that tag. I can use **hg up** to go to any version in the repository.



```
C:\Users\joel\recipes> hg up -r Version-1.0
1 files updated, 0 files merged, 1 files removed, 0 files unresolved

C:\Users\joel\recipes> type guac
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed, minced
...
```

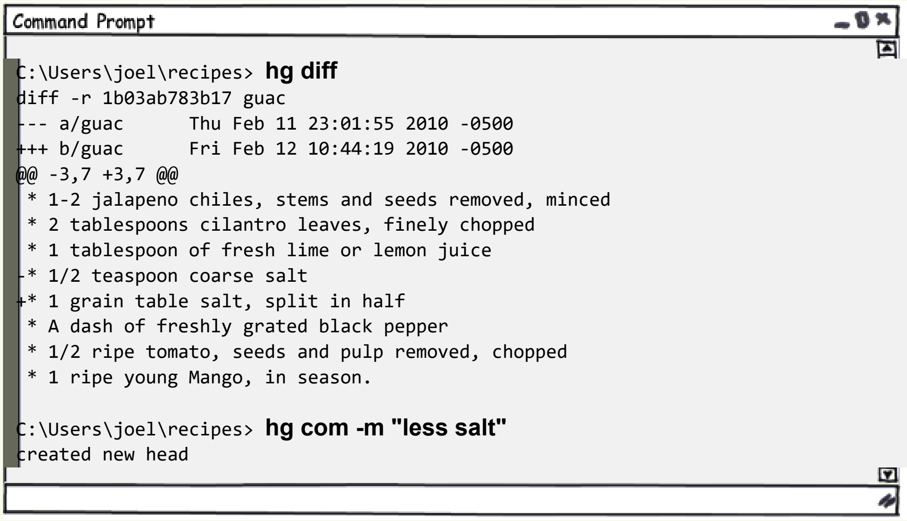
And now I can fix his stupid salt problem:



```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped
* 1 ripe young Mango, in season.
* 1 delicious, yellow BANANA.
```

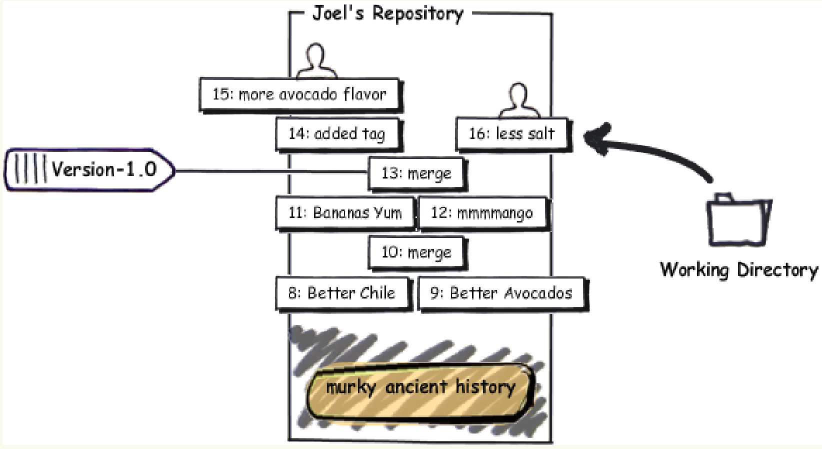
```
* 2 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed,
minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1 grain table salt, split in half
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed,
chopped
* 1 ripe young Mango, in season.
* 1 delicious, yellow BANANA.
```

And:

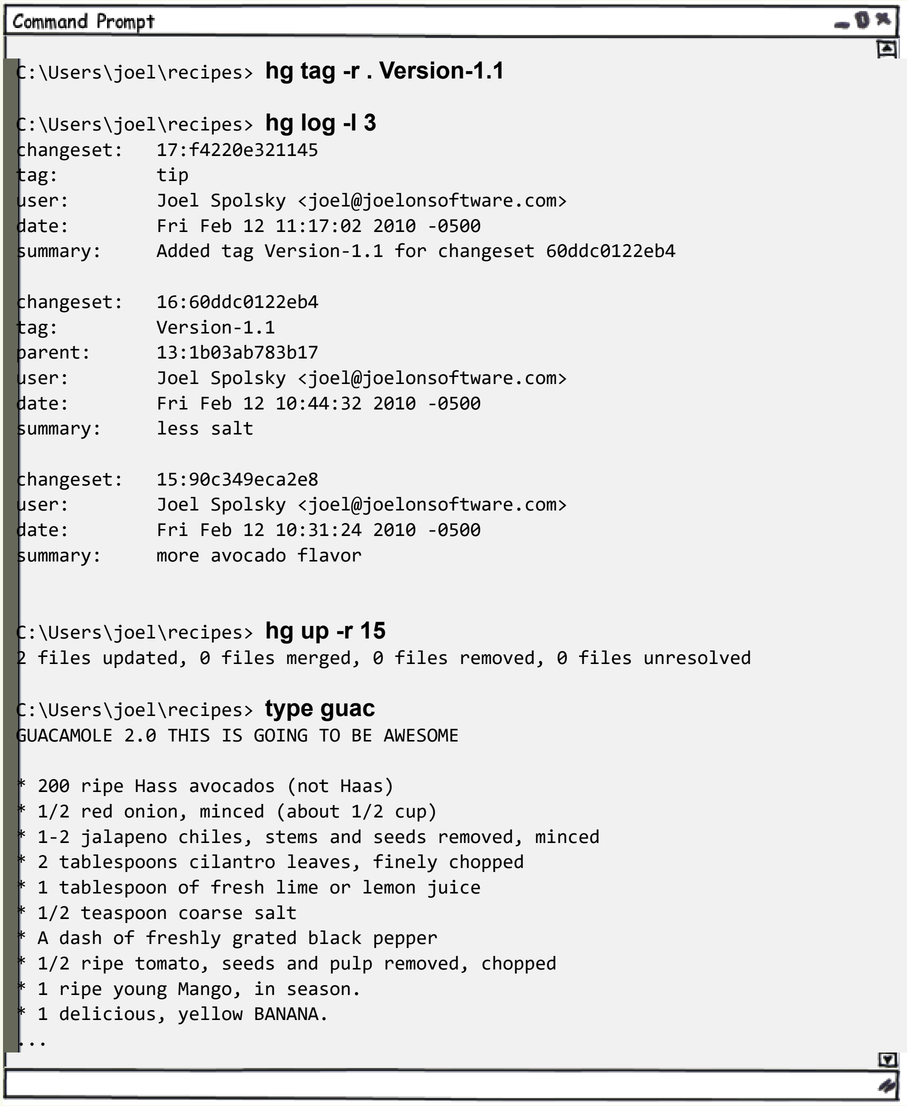


```
C:\Users\joel\recipes> hg diff
diff -r 1b03ab783b17 guac
--- a/guac      Thu Feb 11 23:01:55 2010 -0500
+++ b/guac      Fri Feb 12 10:44:19 2010 -0500
@@ -3,7 +3,7 @@
 * 1-2 jalapeno chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped
 * 1 tablespoon of fresh lime or lemon juice
-* 1/2 teaspoon coarse salt
+* 1 grain table salt, split in half
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
 * 1 ripe young Mango, in season.

C:\Users\joel\recipes> hg com -m "less salt"
created new head
```

Mercurial reminds me that I’ve created a new head. There are two heads now, the 2.0 head I was working on a second ago, and the 1.1 head I just committed.

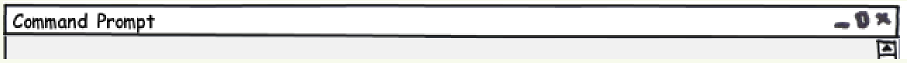


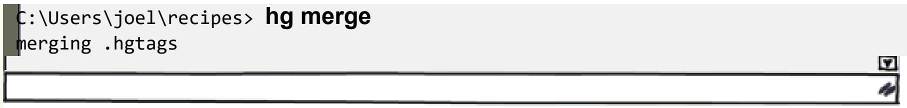
Now I can ship this to my customer, tag it as 1.1, and go back to work on version 2.0.



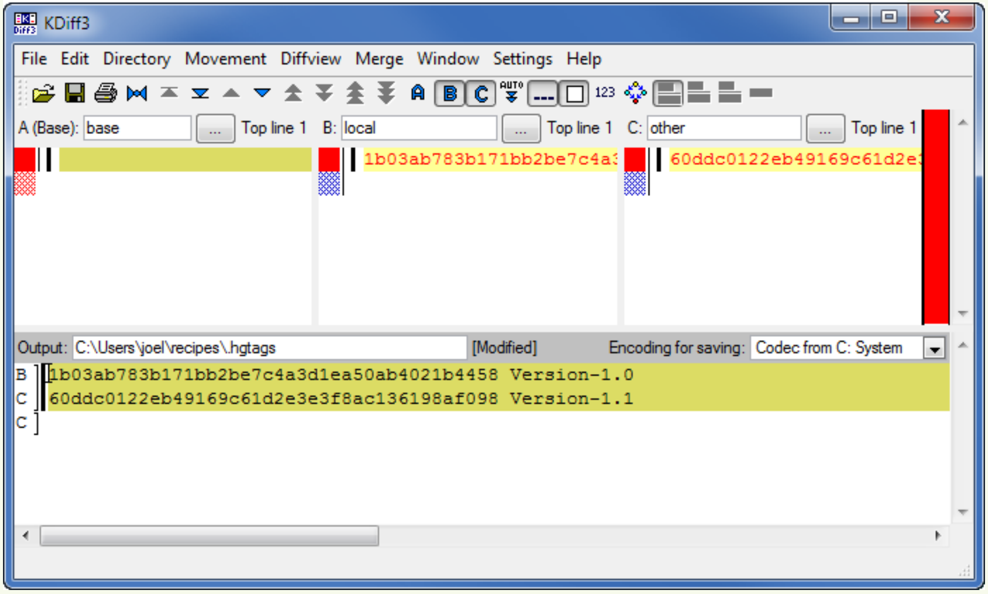
```
C:\Users\joel\recipes> hg tag -r . Version-1.1

C:\Users\joel\recipes> hg log -l 3
changeset:   17:f4220e321145
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 11:17:02 2010 -0500
summary:     Added tag Version-1.1 for changeset 60ddc0122eb4

changeset:   16:60ddc0122eb4
tag:         Version-1.1
parent:      13:1b03ab783b17
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 10:44:32 2010 -0500
summary:     less salt

changeset:   15:90c349eca2e8
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 10:31:24 2010 -0500
summary:     more avocado flavor


C:\Users\joel\recipes> hg up -r 15
2 files updated, 0 files merged, 0 files removed, 0 files unresolved

C:\Users\joel\recipes> type guac
GUACAMOLE 2.0 THIS IS GOING TO BE AWESOME

* 200 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1/2 teaspoon coarse salt
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped
* 1 ripe young Mango, in season.
* 1 delicious, yellow BANANA.
...
```

Only one problem… the salt fix is not here. How do I solve that?





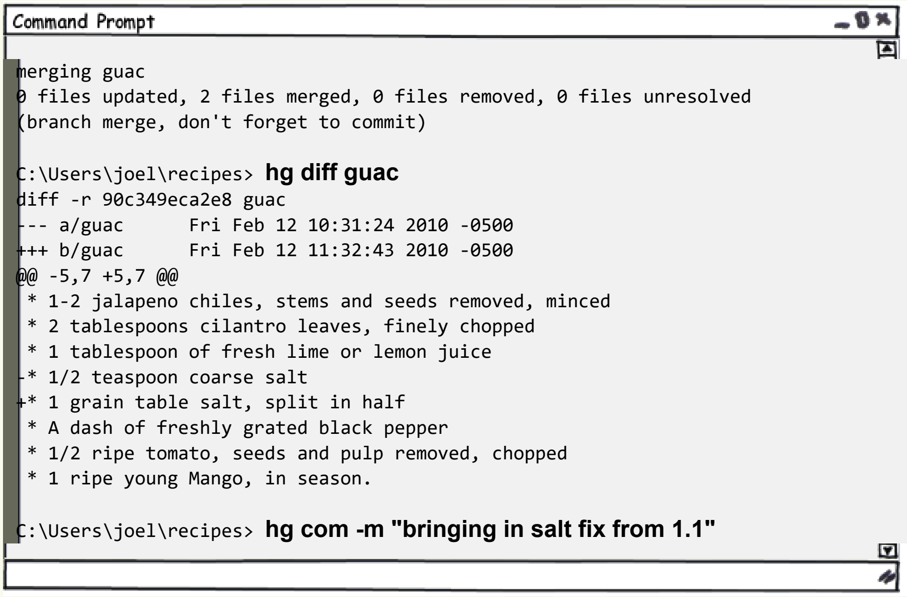
```
C:\Users\joel\recipes> hg merge
merging .hgtags
```



KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A (Base): base
Top line 1
B: local
Top line 1
C: other
Top line 1
Encoding for saving: Codec from C: System

```
1b03ab783b171bb2be7c4a3
60ddc0122eb49169c61d2e3
Output: C:\Users\joel\recipes\.hgtags    [Modified]
B 1b03ab783b171bb2be7c4a3d1ea50ab4021b4458 Version-1.0
C 60ddc0122eb49169c61d2e3e3f8ac136198af098 Version-1.1
C
```

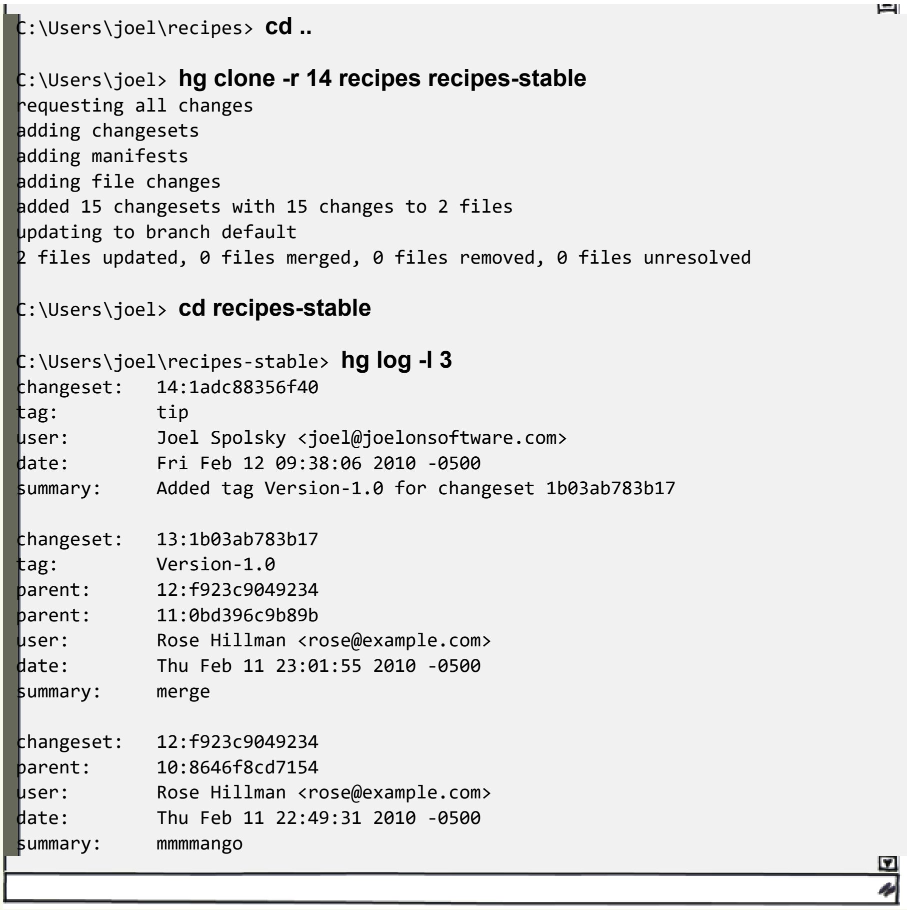
Uh oh. I have to merge the tags. This is an annoying bug in Mercurial. The problem is that tags in Mercurial are just a file, named .hgtags, which is also under version control, so occasionally, you have to manually merge different versions of the .hgtags file. Whenever that happens, what you have to do is easy… always keep BOTH versions of every line in the file.



```
merging guac
0 files updated, 2 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\joel\recipes> hg diff guac
diff -r 90c349eca2e8 guac
--- a/guac      Fri Feb 12 10:31:24 2010 -0500
+++ b/guac      Fri Feb 12 11:32:43 2010 -0500
@@ -5,7 +5,7 @@
 * 1-2 jalapeno chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped
 * 1 tablespoon of fresh lime or lemon juice
-* 1/2 teaspoon coarse salt
+* 1 grain table salt, split in half
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
 * 1 ripe young Mango, in season.

C:\Users\joel\recipes> hg com -m "bringing in salt fix from 1.1"
```

This simple method of going back to old, tagged versions is fine for when you just want to make one little unexpected fix to shipping code. But the truth is, in most software projects, this kind of stuff happens all the time, and Mercurial has a more robust way to deal with it.

So I'm just going to undo all that post-1.0 work, and get the repository back to the way it was when I had just shipped 1.0, and then I can show you the fancy robust way to fix existing customers' bugs while working on a future version at the same time.



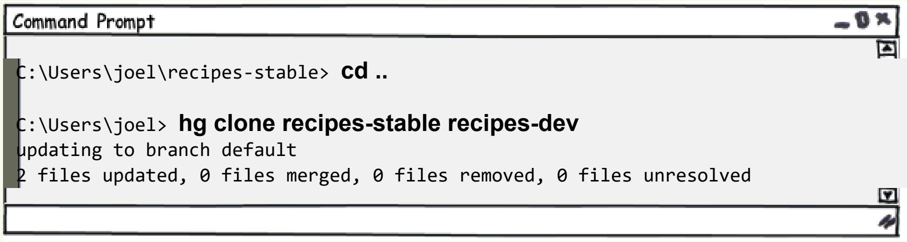
```
C:\Users\joel\recipes> cd ..

C:\Users\joel> hg clone -r 14 recipes recipes-stable
requesting all changes
adding changesets
adding manifests
adding file changes
added 15 changesets with 15 changes to 2 files
updating to branch default
2 files updated, 0 files merged, 0 files removed, 0 files unresolved

C:\Users\joel> cd recipes-stable

C:\Users\joel\recipes-stable> hg log -l 3
changeset:   14:1adc88356f40
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 09:38:06 2010 -0500
summary:     Added tag Version-1.0 for changeset 1b03ab783b17

changeset:   13:1b03ab783b17
tag:         Version-1.0
parent:      12:f923c9049234
parent:      11:0bd396c9b89b
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 23:01:55 2010 -0500
summary:     merge

changeset:   12:f923c9049234
parent:      10:8646f8cd7154
user:        Rose Hillman <rose@example.com>
date:        Thu Feb 11 22:49:31 2010 -0500
summary:     mmmmango
```

The idea is that instead of doing everything in one repository, we’re going to make two repositories, one called **stable** and the other called **dev**.

The **stable** repository holds the last major version of the code that we shipped to customers. Whenever you need to make an urgent bug fix, you do it in **stable**. In this example it’s going to be all the patches to 1.0.

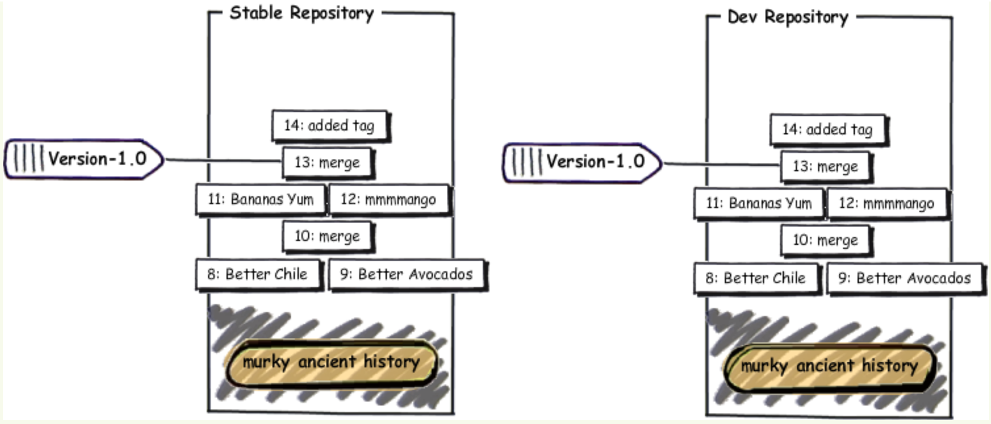
The **dev** repository is where the next major train of development is happening, leading up to 2.0.

As soon as 1.0 ships, I’m going to clone **stable** to **dev**:



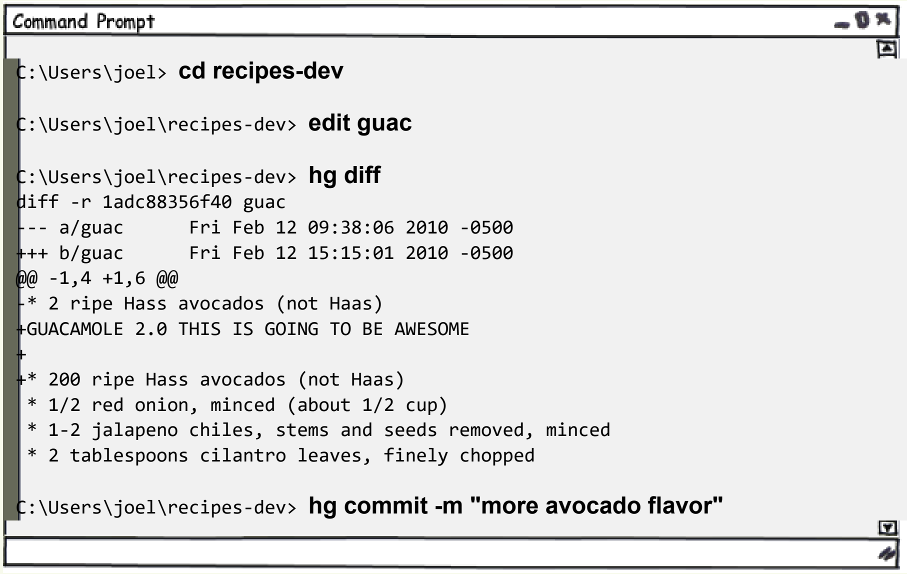
```
C:\Users\joel\recipes-stable> cd ..

C:\Users\joel> hg clone recipes-stable recipes-dev
updating to branch default
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
```

Now I have two identical repositories:



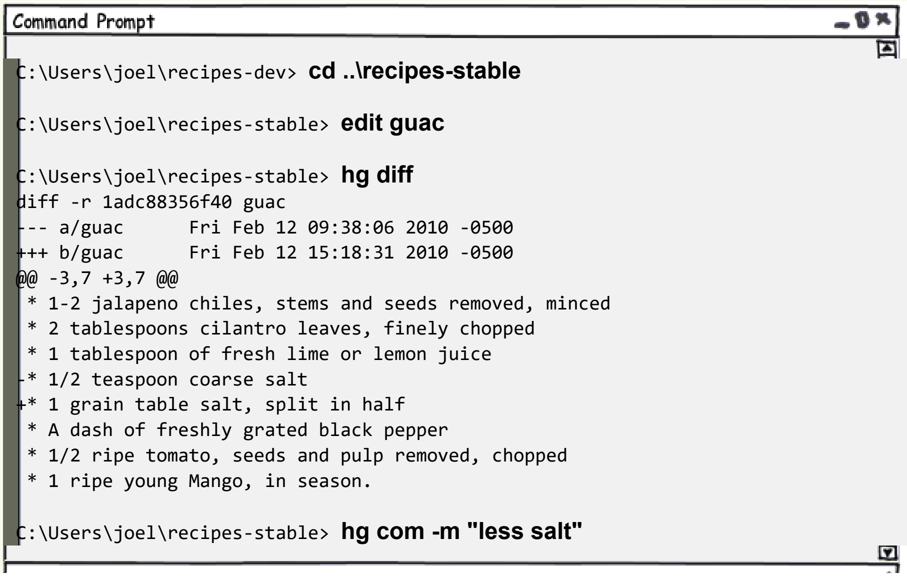
Since the history of both of these repositories is the same up to changeset 14, Mercurial will actually use a file system trick called hard links to avoid making two copies on disk. That makes the **hg clone** operation fast and cheap, so you shouldn't hesitate to make lots and lots and lots of clones.

Now we start working on that guac in the **dev** repository:



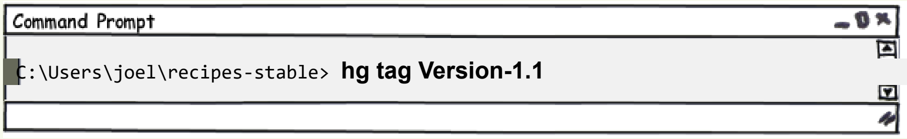
```
C:\Users\joel> cd recipes-dev

C:\Users\joel\recipes-dev> edit guac

C:\Users\joel\recipes-dev> hg diff
diff -r 1adc88356f40 guac
--- a/guac      Fri Feb 12 09:38:06 2010 -0500
+++ b/guac      Fri Feb 12 15:15:01 2010 -0500
@@ -1,4 +1,6 @@
-* 2 ripe Hass avocados (not Haas)
+GUACAMOLE 2.0 THIS IS GOING TO BE AWESOME
+
+* 200 ripe Hass avocados (not Haas)
 * 1/2 red onion, minced (about 1/2 cup)
 * 1-2 jalapeno chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped

C:\Users\joel\recipes-dev> hg commit -m "more avocado flavor"
```

And fix the salt problem in the stable repository:



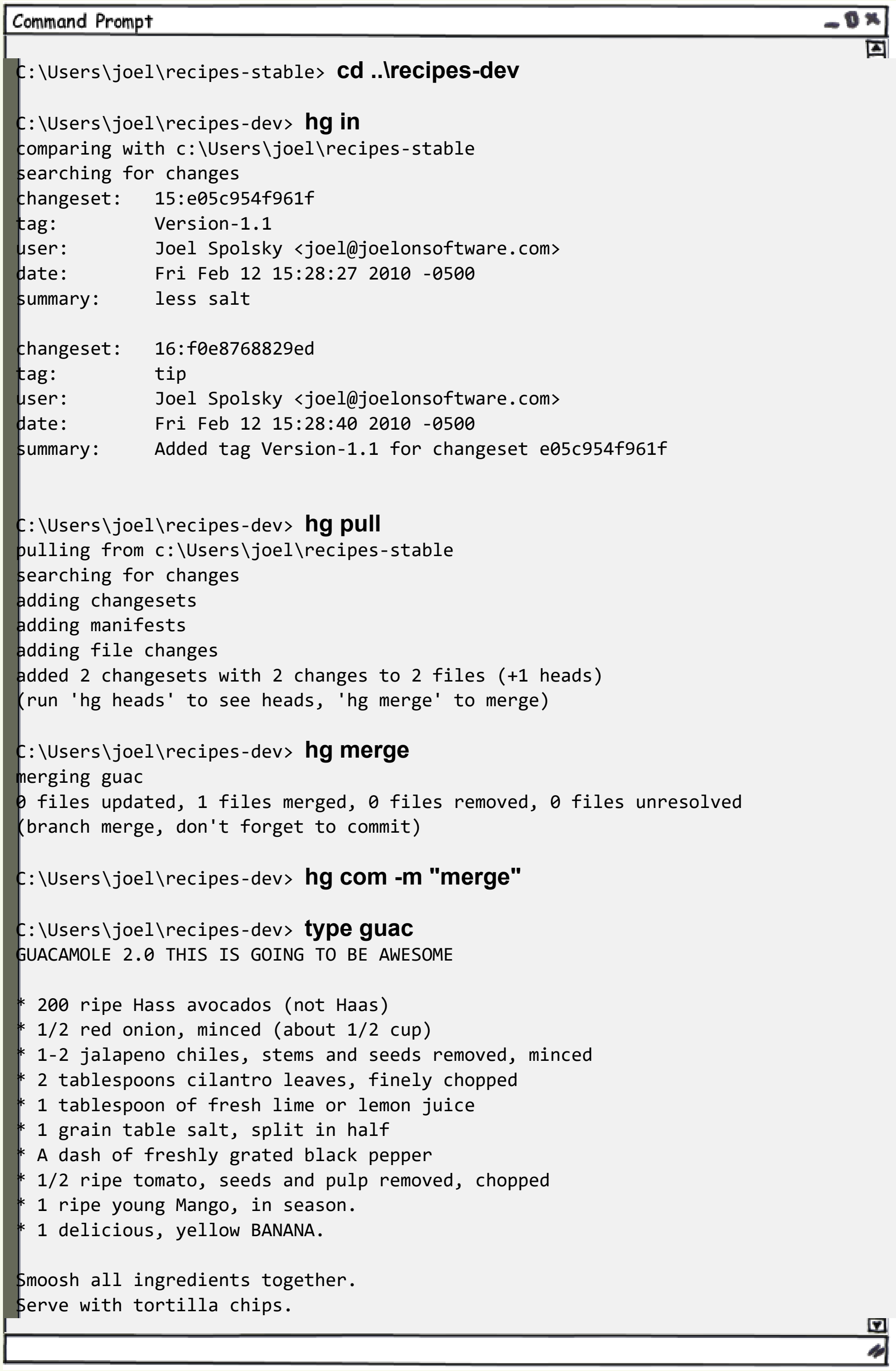
```
C:\Users\joel\recipes-dev> cd ..\recipes-stable

C:\Users\joel\recipes-stable> edit guac

C:\Users\joel\recipes-stable> hg diff
diff -r 1adc88356f40 guac
--- a/guac      Fri Feb 12 09:38:06 2010 -0500
+++ b/guac      Fri Feb 12 15:18:31 2010 -0500
@@ -3,7 +3,7 @@
 * 1-2 jalapeno chiles, stems and seeds removed, minced
 * 2 tablespoons cilantro leaves, finely chopped
 * 1 tablespoon of fresh lime or lemon juice
-* 1/2 teaspoon coarse salt
+* 1 grain table salt, split in half
 * A dash of freshly grated black pepper
 * 1/2 ripe tomato, seeds and pulp removed, chopped
 * 1 ripe young Mango, in season.

C:\Users\joel\recipes-stable> hg com -m "less salt"
```

I’ll tag that and ship it as 1.1:



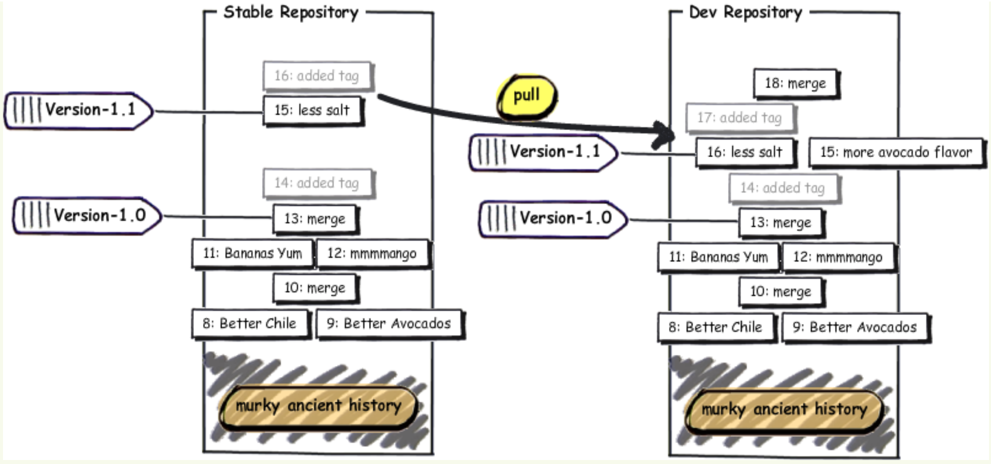
```
C:\Users\joel\recipes-stable> hg tag Version-1.1
```

And now, every once in a while, we pull in the fixes from stable to dev:



```
C:\Users\joel\recipes-stable> cd ..\recipes-dev

C:\Users\joel\recipes-dev> hg in
comparing with c:\Users\joel\recipes-stable
searching for changes
changeset:   15:e05c954f961f
tag:         Version-1.1
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 15:28:27 2010 -0500
summary:     less salt

changeset:   16:f0e8768829ed
tag:         tip
user:        Joel Spolsky <joel@joelonsoftware.com>
date:        Fri Feb 12 15:28:40 2010 -0500
summary:     Added tag Version-1.1 for changeset e05c954f961f


C:\Users\joel\recipes-dev> hg pull
pulling from c:\Users\joel\recipes-stable
searching for changes
adding changesets
adding manifests
adding file changes
added 2 changesets with 2 changes to 2 files (+1 heads)
(run 'hg heads' to see heads, 'hg merge' to merge)

C:\Users\joel\recipes-dev> hg merge
merging guac
0 files updated, 1 files merged, 0 files removed, 0 files unresolved
(branch merge, don't forget to commit)

C:\Users\joel\recipes-dev> hg com -m "merge"

C:\Users\joel\recipes-dev> type guac
GUACAMOLE 2.0 THIS IS GOING TO BE AWESOME

* 200 ripe Hass avocados (not Haas)
* 1/2 red onion, minced (about 1/2 cup)
* 1-2 jalapeno chiles, stems and seeds removed, minced
* 2 tablespoons cilantro leaves, finely chopped
* 1 tablespoon of fresh lime or lemon juice
* 1 grain table salt, split in half
* A dash of freshly grated black pepper
* 1/2 ripe tomato, seeds and pulp removed, chopped
* 1 ripe young Mango, in season.
* 1 delicious, yellow BANANA.

Smoosh all ingredients together.
Serve with tortilla chips.
```

Here’s what we’ve been doing:

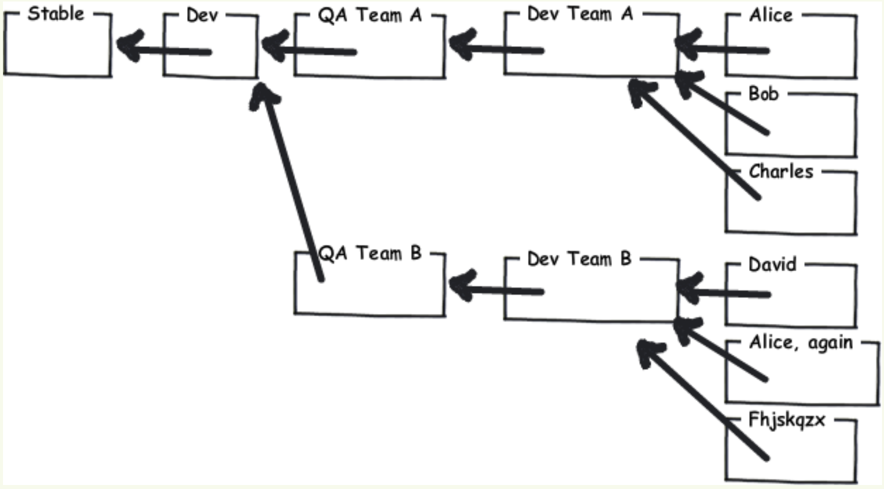


And, if you can understand *that* crazy graph, you're well on your way to understanding Mercurial. The gist of it is that eventually the stable repository only has the bug fixes, while the dev repository has new code and bug fixes merged in.

There are other ways you can use multiple repositories.

- You can set up a team repository where a few people work on a feature together. When they are finished and the feature works well, you push the changes from the team repository to the main development repository, and then everyone else sees it.
- You can set up a QA repository for the testers. Instead of pushing into the main repository, you push into the QA repository where your code is tested. Once the testers approve it, you push from the QA repository into the main development repository. That way the main repository always contains tested code.
- Since each developer has their own repository, you can pull an experimental changeset from your friend to test it, without inflicting that change on the rest of the team.

In large, complicated organizations, you can combine these techniques, setting up a bunch of repositories that pull from each other. As each feature goes through testing and integration, it gets pulled up higher and higher in the tree until eventually the new code is in the main shipping repository where customers can get it:



Test yourself

Here are the things you should know how to do after reading this tutorial:

1. Tag old versions and go back to them
2. Organize your team with "stable" and "dev" repositories

Well, somehow we've reached the end of this tutorial. I haven't even come *close* to covering everything in Mercurial, but there are plenty of resources to go more in depth. There's a book that covers everything in utter and complete detail. And if you have any questions, I'd like to invite you to the Kiln Knowledge Exchange (it's like StackOverflow, but just for Kiln, and Mercurial questions are more than welcome there.)

« Home

To learn more, **read the book** or hang out at the **Kiln Knowledge Exchange**.

Any questions?

If you have any questions about the material in this tutorial, no matter how newbie, ask them at the Kiln Knowledge Exchange.

About the author.

Joel Spolsky is the founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they're delighted. Fog Creek makes FogBugz, Kiln, and Fog Creek Copilot. Joel's blog Joel on Software is read by programmers everywhere.