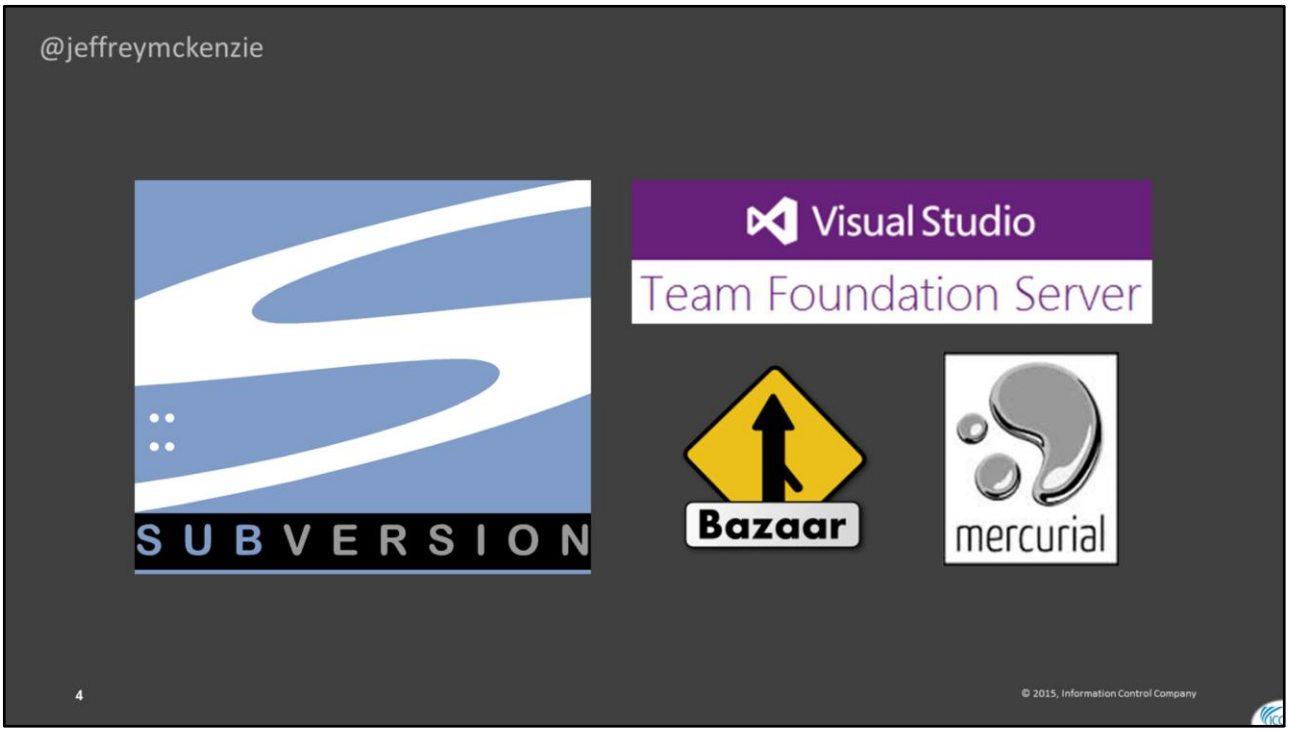
Good morning everyone – My name is Jeff McKenzie
First, I would like to thank Pittsburgh Tech Fest
For having me today, and thank all of you for attending –
Lots of great sessions going on, and I know you all
have a choice when it comes to which session you attend.

And I work as a managing architect for ICC,
a consulting firm based in Columbus ohio.
We help companies solve business problems
Through technology, with offerings in
Application Development, for .NET,
Java, open source, and offerings in
Business Analytics, Core Infrastructure,
As well as digital marketing.
ICC is the largest privately held IT consulting firm
In Central Ohio, and has clients nationally
As well as throughout The Great Lakes Region.

Today we are going to be taking a look a the Git version control system,
but we are going to approach it in a slightly different way than you might otherwise.

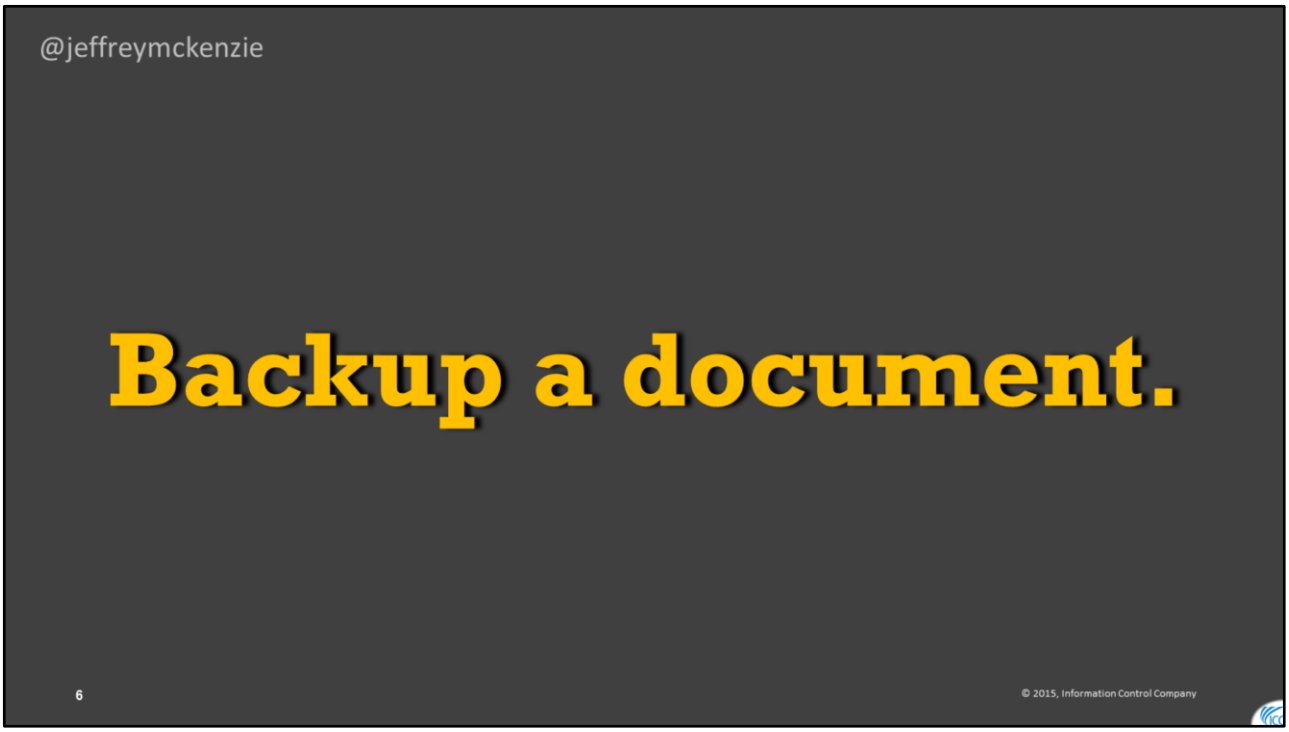
Today, if you look around on the internet and try to learn about Git,
you're going to find a lot of tutorials that start you off with a series of commands,

Such as – here's how to start git, or here's how you add a file.



and that might be fine if you are familiar with version control,
and have used many different kinds of systems,
so you have something to compare git to.
But what if you haven't? What if you've only ever used one system,
or are completely new to version control?

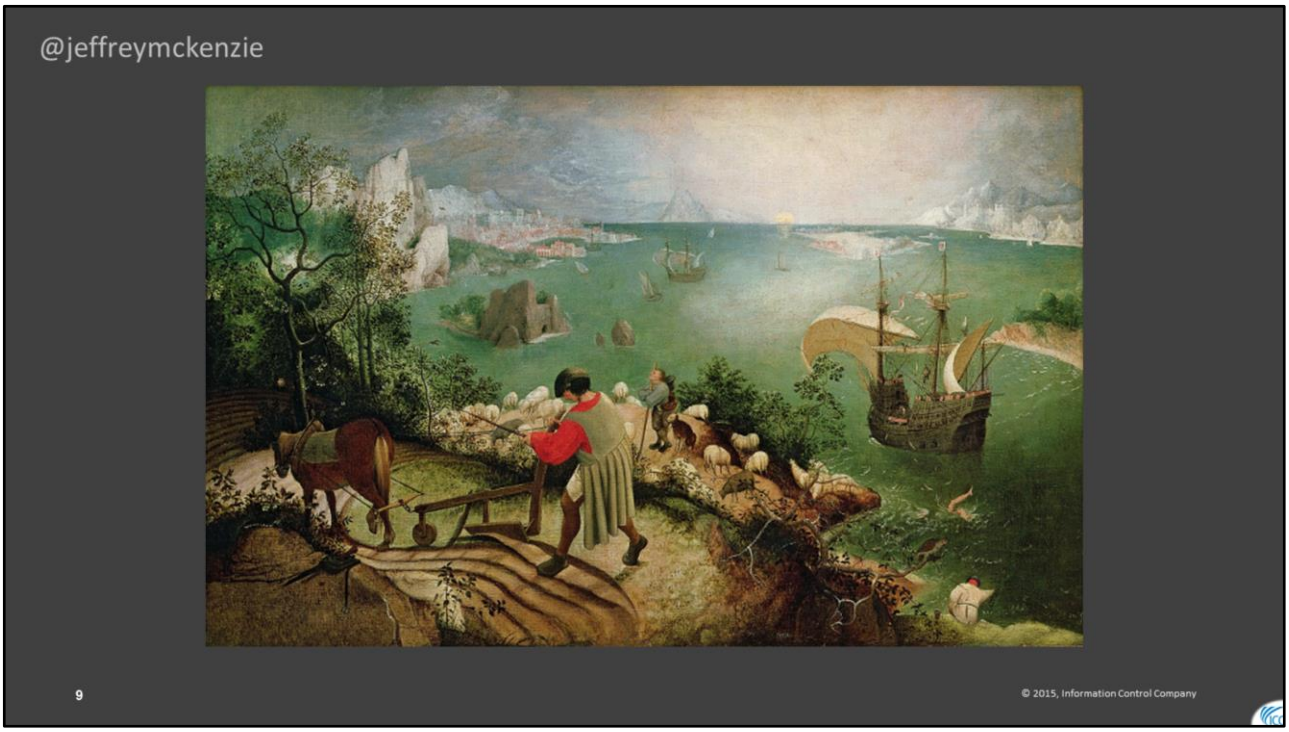
Before we jump into the specifics,
I want to ask: what is version control?
Can anyone give me some examples of version control?
[discussion]
My own definition of version control is pretty broad –
In my opinion, there are 3 fundamental attributes
that are required for version control.



First, the ability to store and backup a document.

Second, the ability to track who made the change and when it was made.

The combination of those first two items enables the third:
the ability to revert to a specific version. By having a backup document with a date on it,
I'm able to go back to an older version in case of a mistake or data loss.
We will revisit these with an example later.



Speaking of version control, I was in the library the other day,
and in the new books section I ran across the history of an ancient civilization.
It was about a place called Gitlandia, and it just so happens
that the story of its founding and growth is strikingly similar
to the different models of source control systems.
So before we get to Git itself, I thought it might be instructive
to take a quick trip to this strange land.

[image: Pieter Brueghel the Elder (1526/1530–1569) [Public domain], via Wikimedia Commons]

@jeffreymckenzie

10

Now Gitlandia didn't get its name until much later in history –
it started life as a small city in a beautiful river valley near the sea.
The place was called Hackistan.
Even though it was small, it had a reputation for having
some of the finest art in the entire region.

[image: By J. M. W. Turner [Public domain, GFDL
(http://www.gnu.org/copyleft/fdl.html) or CC-BY-SA-3.0
(http://creativecommons.org/licenses/by-sa/3.0/)], via Wikimedia Commons]

@jeffreymckenzie

11

Merchants from far-away lands
would travel to this place
to purchase many things.

But some of the best were its exquisite wooden horses.

[image:]

@jeffreymckenzie

12

And the person responsible for making these horses
was one especially talented artist named Marius Maximus.
But most people in Hackistan just called him Max.

[image:]

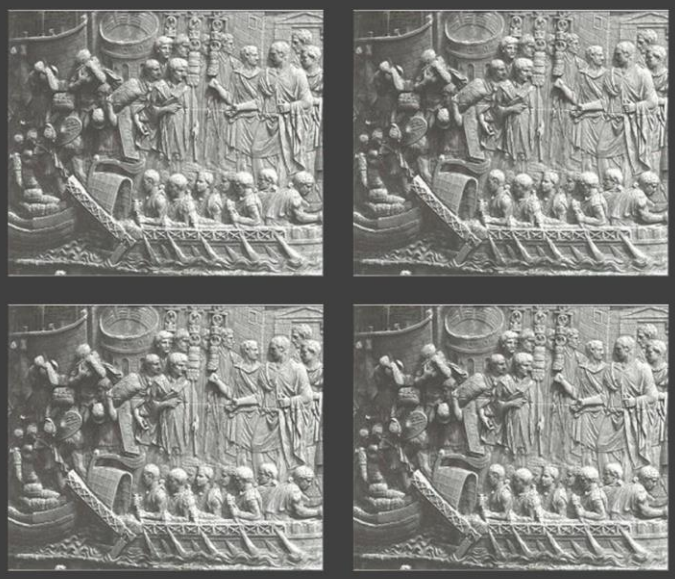
@jeffreymckenzie

13

For years he had been building these horses
using only the knowledge in his head –
from memory, without writing anything down.

[image:]

@jeffreymckenzie

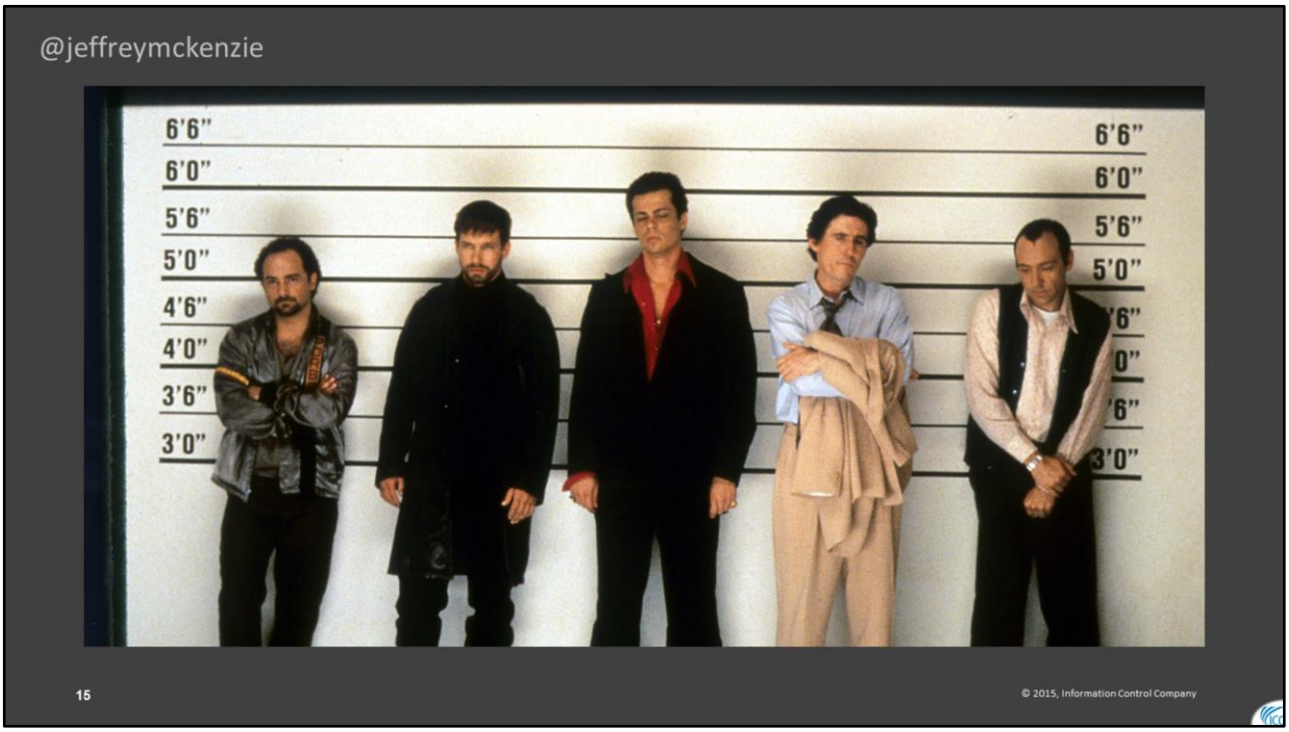


14

And for a while, he did very well by himself, and demand from the merchants increased.

Word of Max's accomplishments spread to the rulers of Hackistan…

[image:]



Naturally the rulers got a cut of the wooden horse sales through taxes.

And the increased taxes meant that the rulers were pleased – well –

As pleased as rulers can get – these are their happy faces.

Because they wanted more of this success,
Max was asked by the rulers of Hackistan to bring on his first apprentice.

[image:]

His name was Regillus Aurelieus, but people
generally just called him Reggie.

Unfortunately, Reggie was not as photogenic as Max.
There are very few likenesses of him, and honestly,
this one is the best we have,
So you can imagine what the others look like.

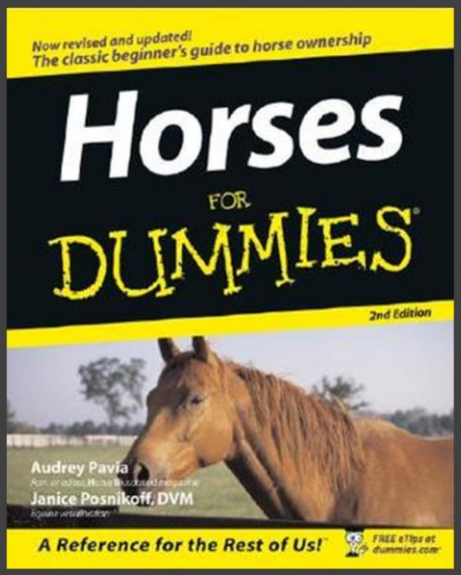
But adding a person presented a problem.

[image:]

What had worked in the beginning, with Max working alone,
clearly would not work now.
Max was busy most of the time actually creating the horses.
So the rulers of Hackistan made a decision
to write down the horse-making instructions in a book.

[image:]

@jeffreymckenzie



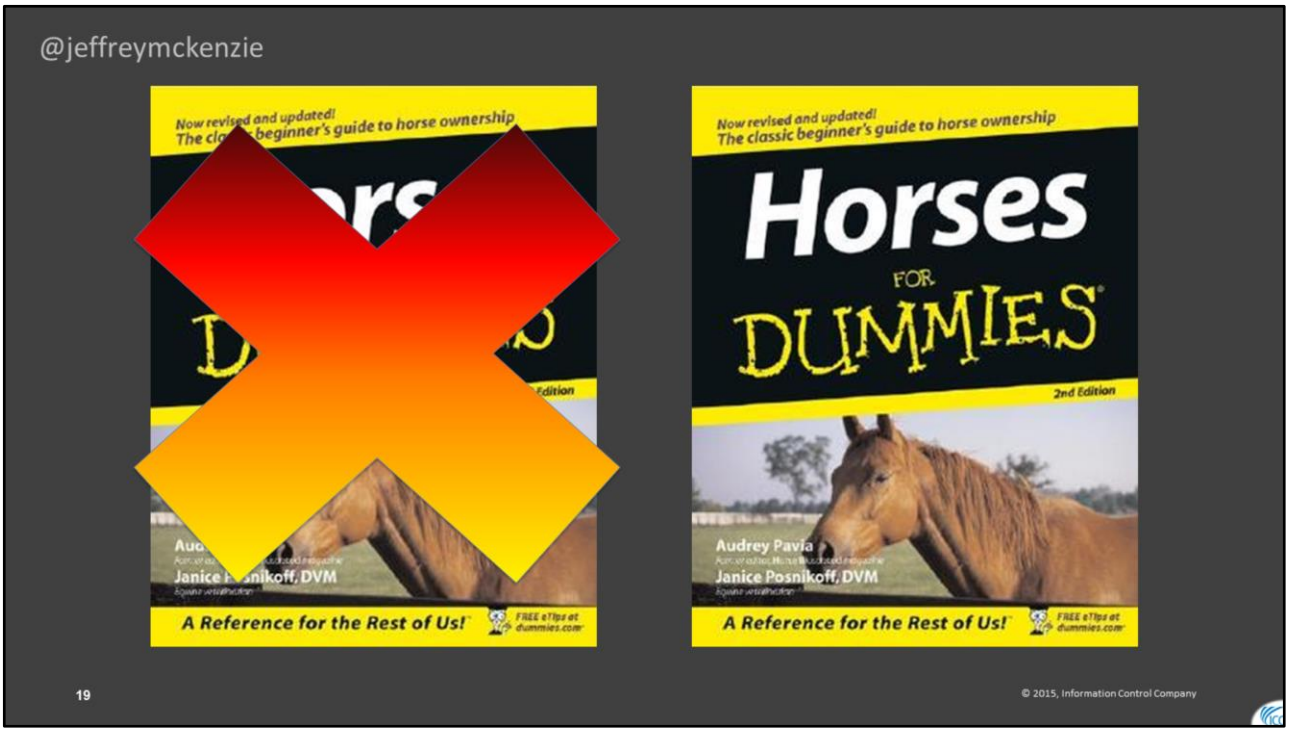
18

We're not exactly sure what that book looked like,
but scholars have chosen this as the most
likely representation of its appearance.
Since Max was the "big ideas" guy.
Reggie would be responsible for writing the book,
using the knowledge passed down from Max.

And after some time, Reggie had finished his book.
And he put it in the library.

As time went on, both Max and Reggie would consult the book,
since it would become the official instructions.

[image:]



As changes were made to the horse design over time, a new book

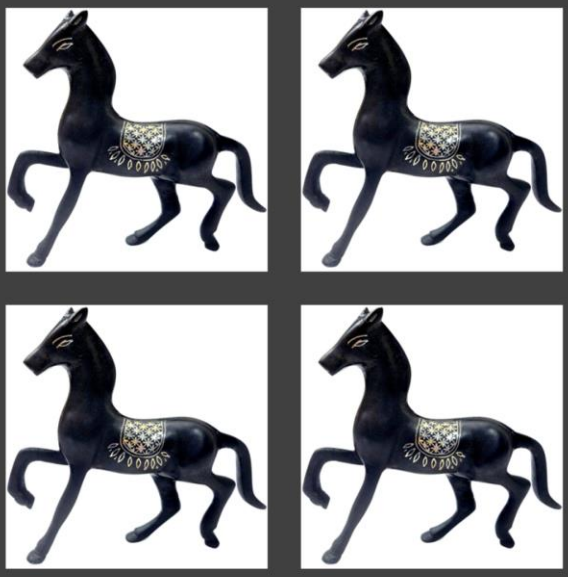
would be written, and the old one would be thrown away.

One day Max discovered a new way to paint the horses, to make them last longer.

As usual, he dictated the change to Reggie, and Reggie replaced the book.

[image:]

@jeffreymckenzie



20

As they completed the next batch of horses and shipped them out to their customers,

they discovered that all was not well.

Reggie had made a mistake in the latest copy of the book, and all of the horses started looking like this….

[image:]

@jeffreymckenzie

21

When left outside, the wooden horses became essentially high-priced chia pets.

Customers were furious, and demanded refunds.

Reggie and Max scrambled to find the previous version of the book..

[image:]

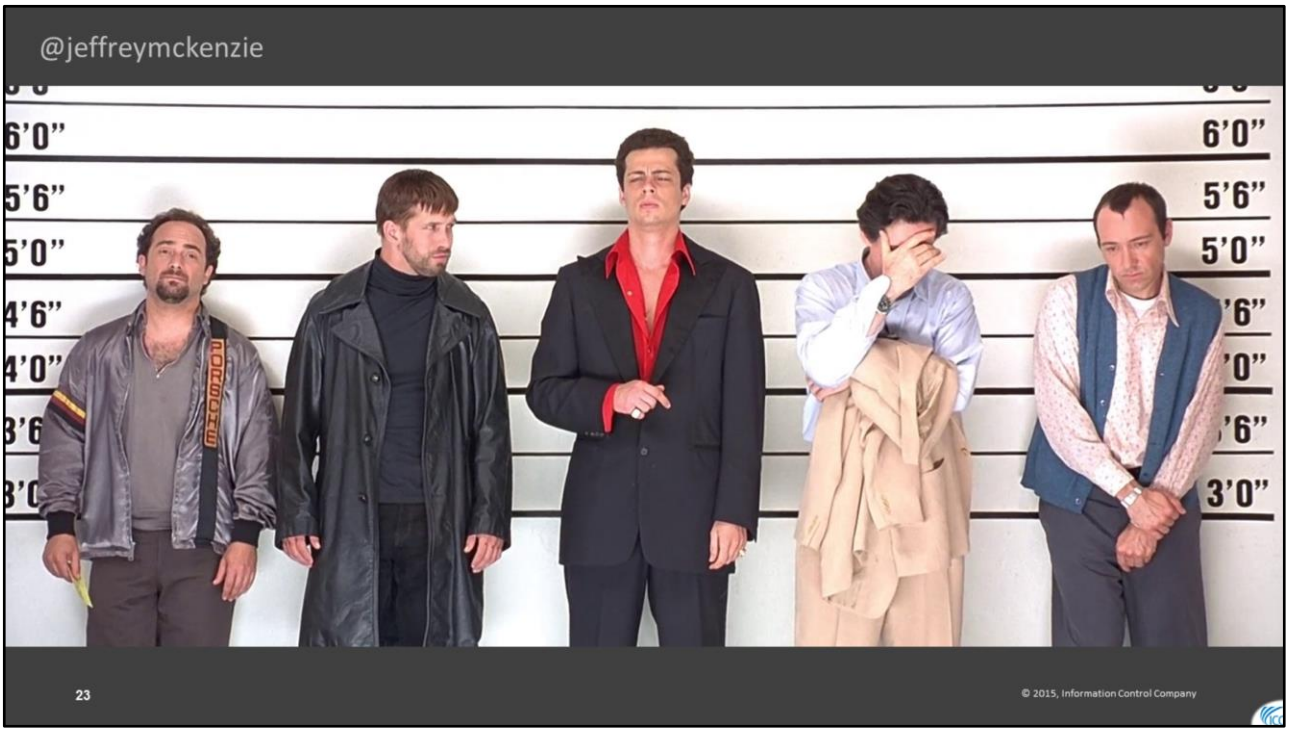
@jeffreymckenzie

22

And after much hard work,
they managed to find the previous instructions
and fix the error that ruined the horses.

So they were back in business.
But the rulers were not happy.

[image:]



[These are their unhappy faces.]

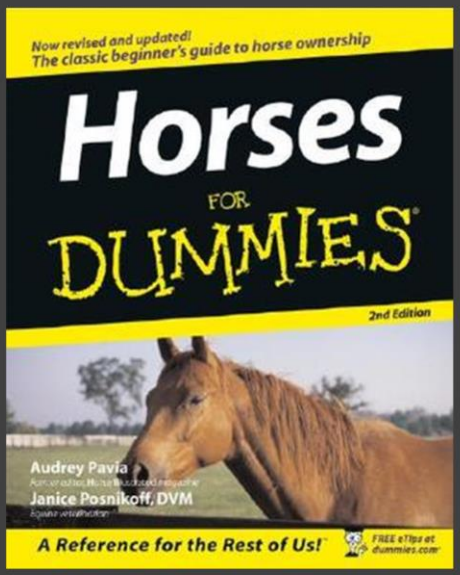
It was simply too close of a call for them,
and they didn't want to have to sell some of their
8 horsepower chariots and fancy aqueduct systems,
not when the money was getting good.

So the rulers decreed that a system be put into place
to prevent such an error from happening again.

Instead of throwing the old book away,
they would simply keep the old book,
and give the new book a different title.

[image:]

@jeffreymckenzie



_version2

24

Something like this – Horses version 2.

The process started out just fine in the beginning….

[image:]

Versions were going on in an orderly fashion,
if a little unorthodox.

But then things took a strange turn…

[image:]

[by the way, has anybody here worked on a version control system like this, or are using one currently?]

In some books they wanted to keep parts of the instructions, even though they weren't yet finished with the changes.

You had to know which versions to take parts from in order to make a complete horse.

Eventually things started to fall apart.
There was no one true version of the instructions, and Max and Reggie Started making horses that looked kind of like this….

[image:]

@jeffreymckenzie

27

And that was pretty much the end of it.
The problems were too many and too complex to solve.

People demanded refunds until the money ran out –
then people wanted new horses, but
Max and Reggie had no way to make them.

The citizens of Hackistan were furious,
and so were the merchants and their customers.

People from the surrounding
towns and villages stormed the city….

[image:]

@jeffreymckenzie

28

They looted the market, and set fire to every building they could find.

The city burned for days until this was all that was left…..

[image:]

@jeffreymckenzie

29

Now, as people who make software,
we'd like to avoid the mistakes of Hackistan
so that our projects don't end
with hordes of dissatisfied customers
carrying pitchforks and torches.

[image:]

@jeffreymckenzie

Part I Hackistan

Manual Version Control

30

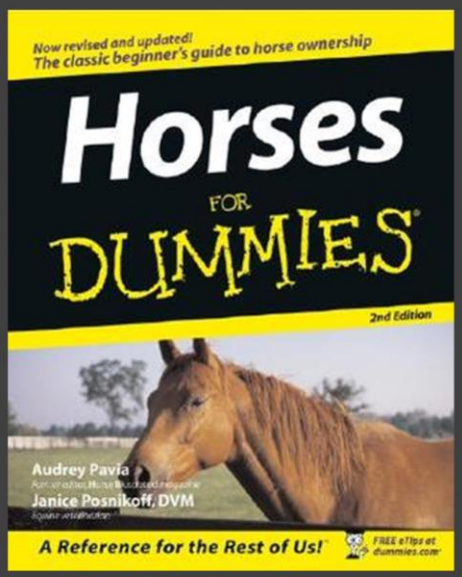
In the story of Hackistan is really a picture of manual, or ad hoc version control – it's not automated, and not really a systematic process.

So if we consider the horses book
to be the document under version control….

[image:]

@jeffreymckenzie

Backup a document?



31

…then how does it fit in with the 3 concepts of version control we discussed earlier?

When did they have a backup of the instructions [creation of the book, keeping versions]?

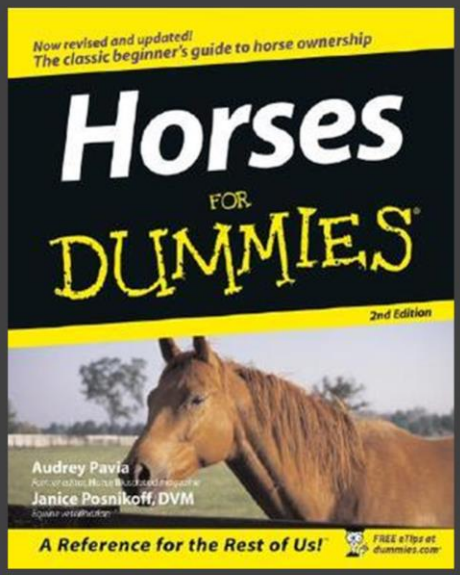
Were they able to track who changed it and when? [sort of, it was possible that they included notations]
Were they able to revert to a specific version? [I think they did initially, but in the end when they needed to consult several books, that became almost impossible to do]

So I think we can say that at one point they had a functioning version control process, but it broke down.

[image:]

@jeffreymckenzie

Track changes: who & when?



32

Were they able to track who changed it and when?
[sort of, it was possible that they included notations]

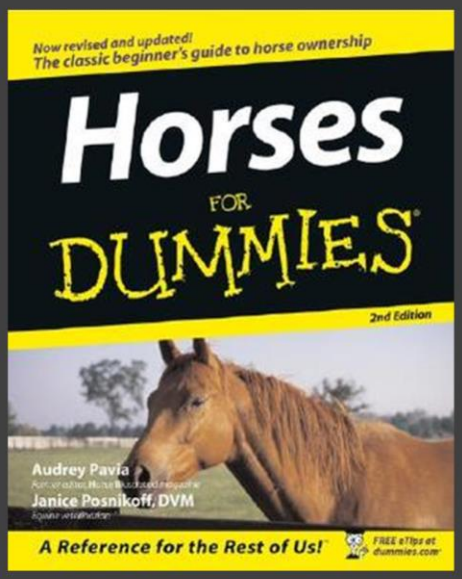
Were they able to revert to a specific version? [I think they did initially, but in the end when they needed to consult several books, that became almost impossible to do]

So I think we can say that at one point they had a functioning version control process, but it broke down.

[image:]

@jeffreymckenzie

Revert to a specific version?



33

Were they able to revert to a specific version?
[I think they did initially, but in the end when
they needed to consult several books, that became almost impossible to do]

So I think we can say that at one point they had a functioning version control process, but it broke down.

[image:]

Based on what we learned in Hackistan,
No matter what you are working on,
Horse instructions, source code,
documentation, graphic design,
There's an easy answer to this question. [CLICK]
Yes, you need version control.

However, the more difficult question is….

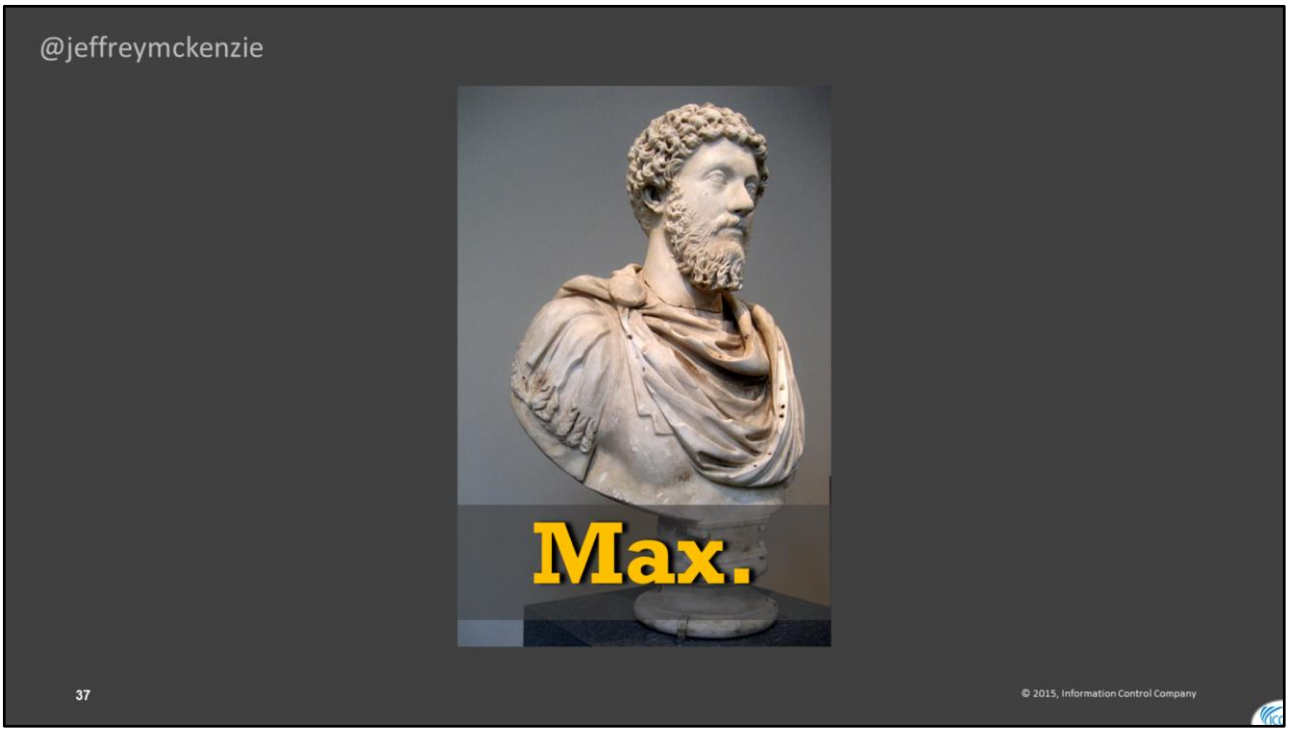
[image:]

Do I need a version control system?
Can anyone tell me the difference between version control
And a version control system?
Sometimes we use the terms interchangeably
But there is a difference.
When you examine the history of hackistan, [CLICK]
I think it becomes clear that as the complexity increases,
the need for a version control system increases,
And that complexity can be either the number of users you have,
or the number of features you have.

Version control systems are very nice to have, but they do incur overhead – it's not free.
You have to install it, configure it, monitor it
And keep it healthy and current.
So you have to make a decision.

In the beginning, when Max was working by himself…

[image:]



his work product was consistent, and didn't change a lot,
so he probably didn't need a version control system

But as the process became more complex,
Max and Reggie needed something more,
and this brings us to the second part of the story.

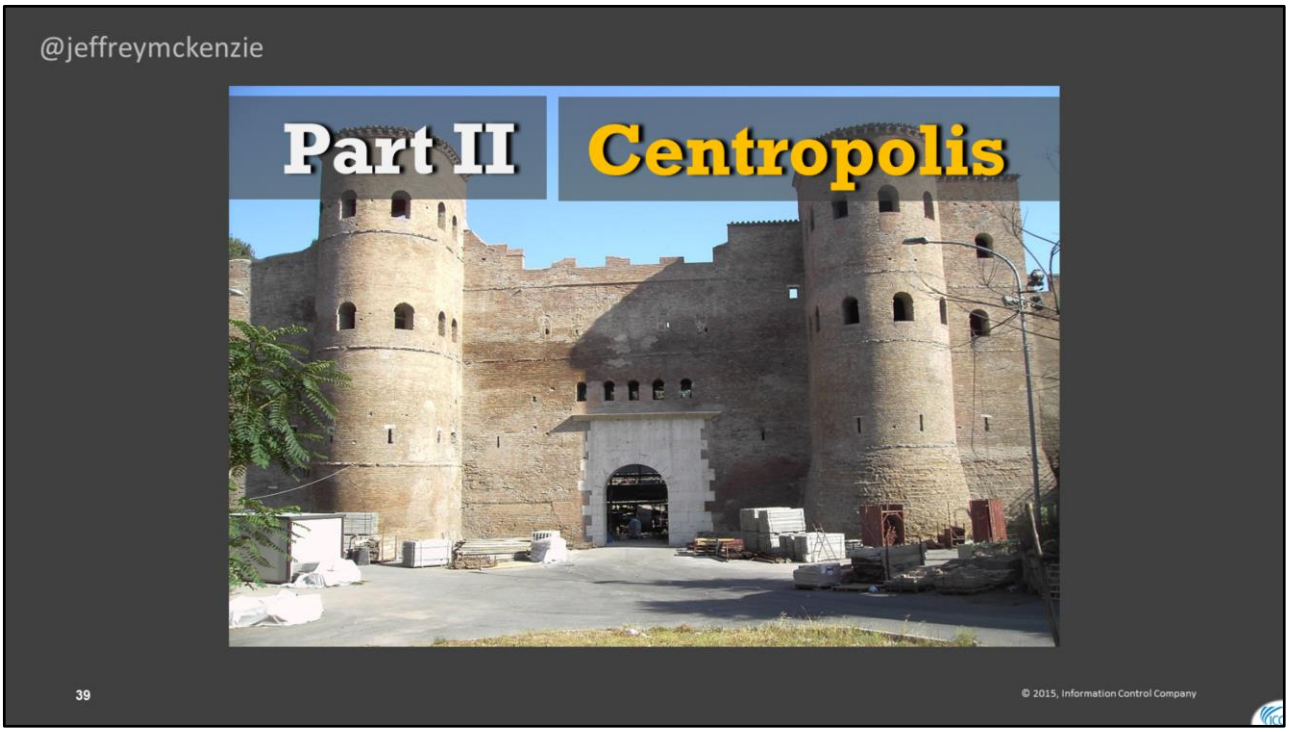
[image:]

Somehow, our cast of characters from Hackistan were able to escape unharmed from the devastation.

Max, Reggie and the rulers managed to collect many of the books from the library before it was burned down.

Together they formed a new city, called Centropolis….

[image:]



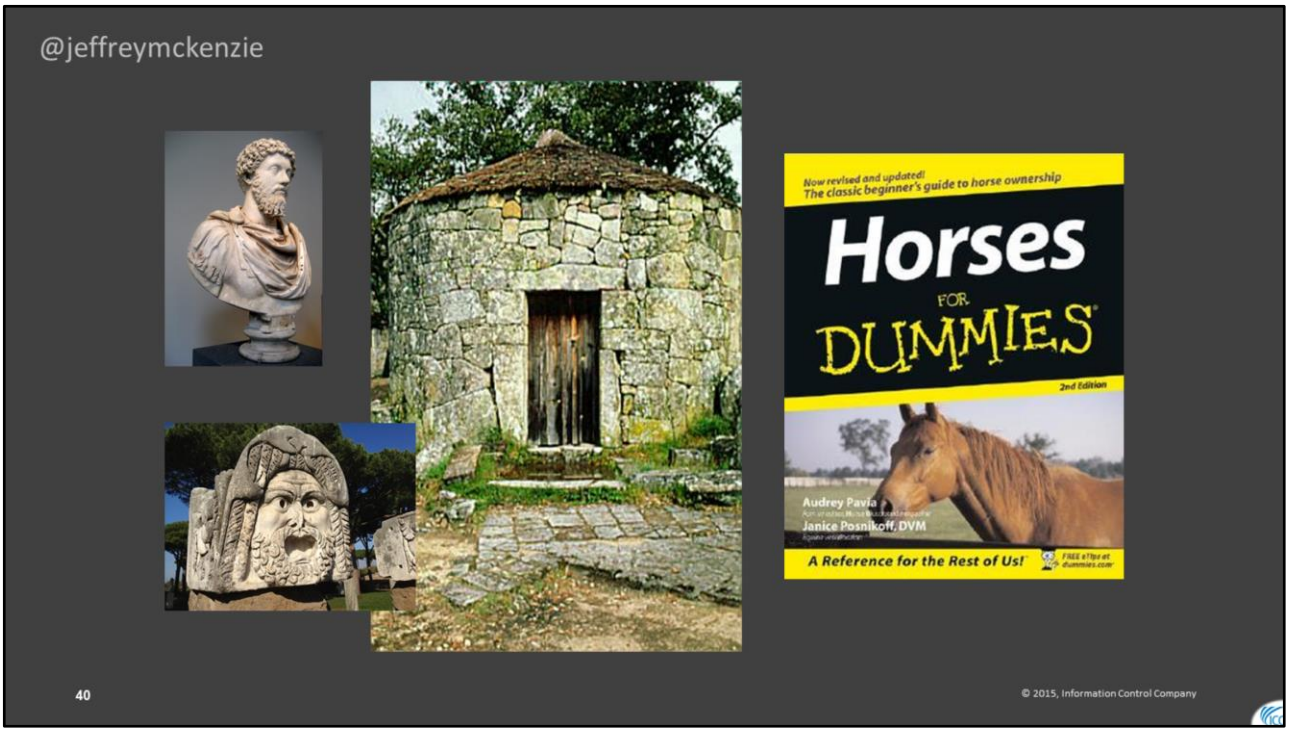
In order to finance the construction of this new city, the rulers had to restart the wooden horse trade.

This time, they wanted to be absolutely sure that their system for version control of the horse-making instructions was solid.

They could not afford another round of looting, pillaging, burning, and so forth.

The rulers surrounded the central library with walls, and put tight controls on access to the library.

[image:]



Max and Reggie started their work in a building like this one,
using a copy of the horse instructions they obtained from the central library.

Whenever they needed to make a change to the instructions,
they would note their changes in the book,
and then they would call for a messenger.

[image:]

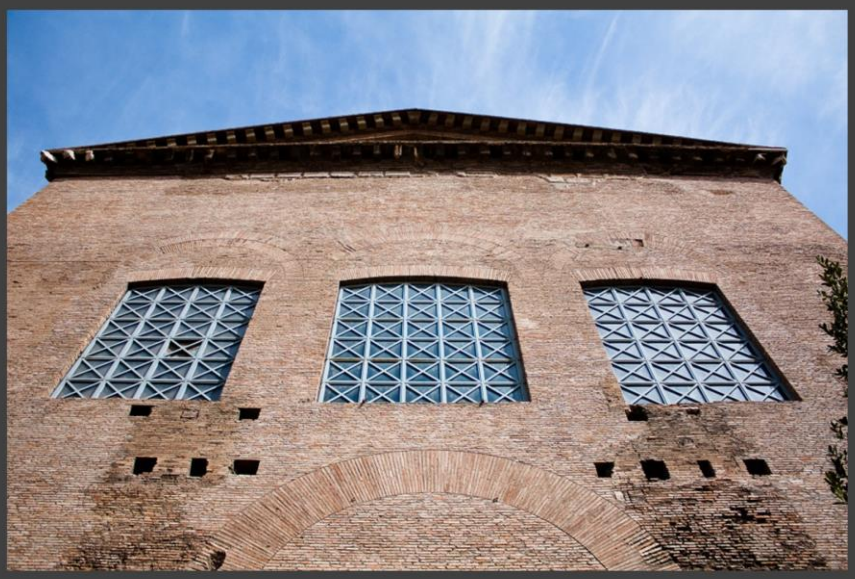
@jeffreymckenzie

41

The messenger would take the edited copy of the book…

[image:]

@jeffreymckenzie

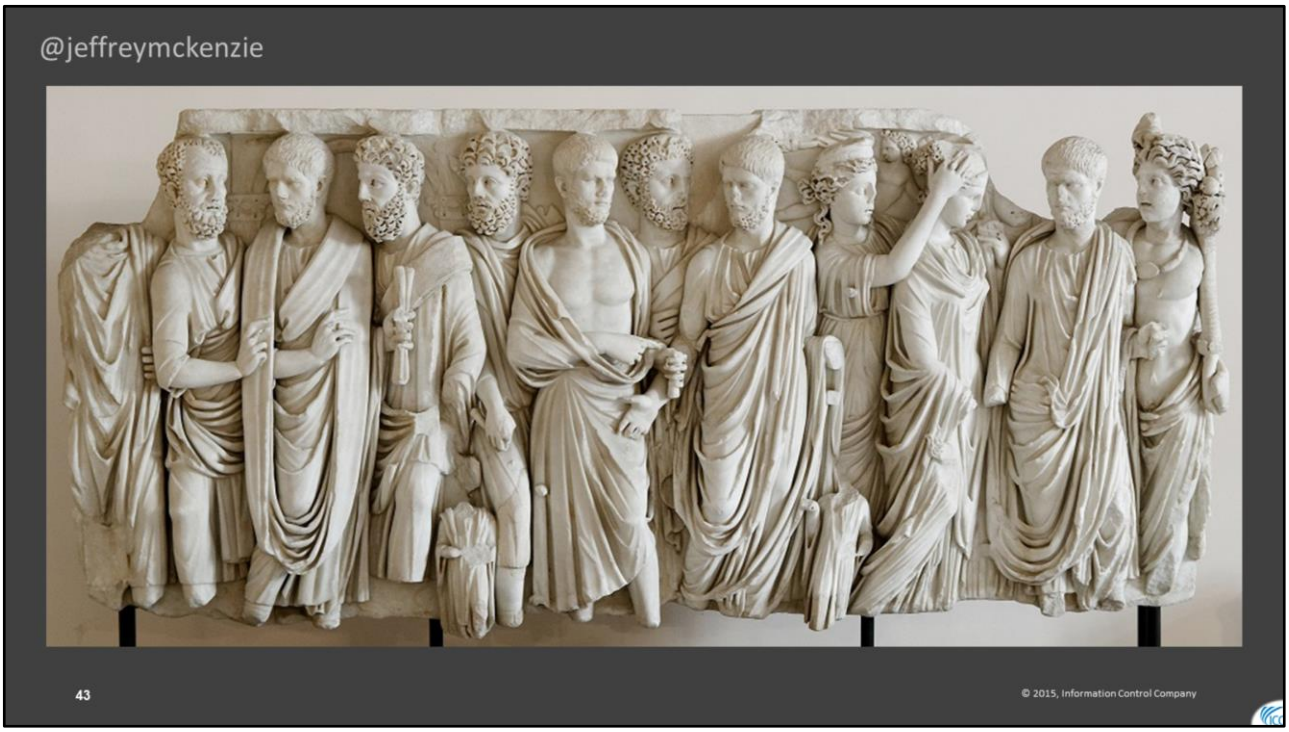


42

back to the central library.

And within the central library was…

[image:]



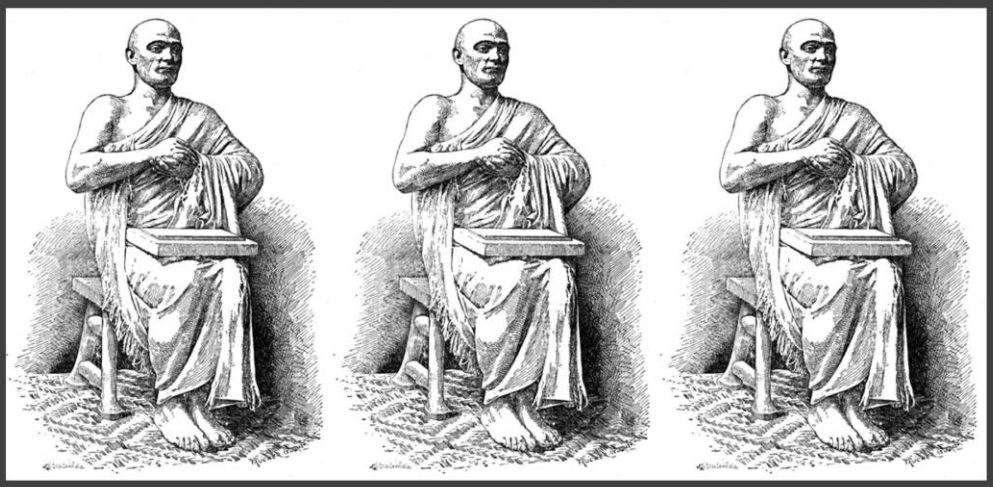
a group of special advisors, a sort of council.
And it was the job of the council to
review and approve changes to the book.

This provided a great deal of control and
oversight over the changes made by the workers.
Because as we all know, there is no problem so large
that can't be solved by an additional layer of bureaucracy.

When these changes were approved,
they were passed along to a team of scribes
who would make actual changes to the book.

[image:]

@jeffreymckenzie



44

This proved to be a lot faster, and more efficient –
instead of having Max and Reggie copy the book themselves,

They could continue to build the wooden horses
while the copying took place elsewhere.

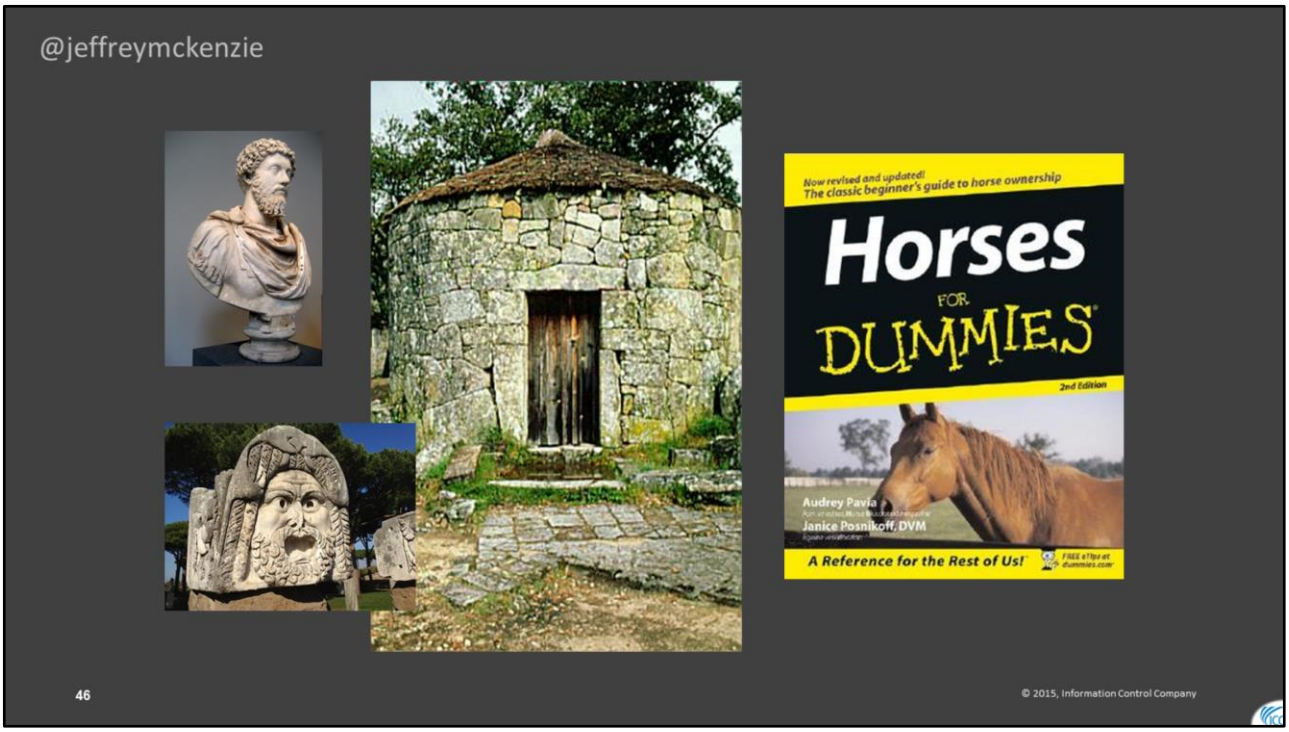
[image:]

@jeffreymckenzie

45

Then the messenger would take the new copy of the book back to Max and Reggie in their workshop.

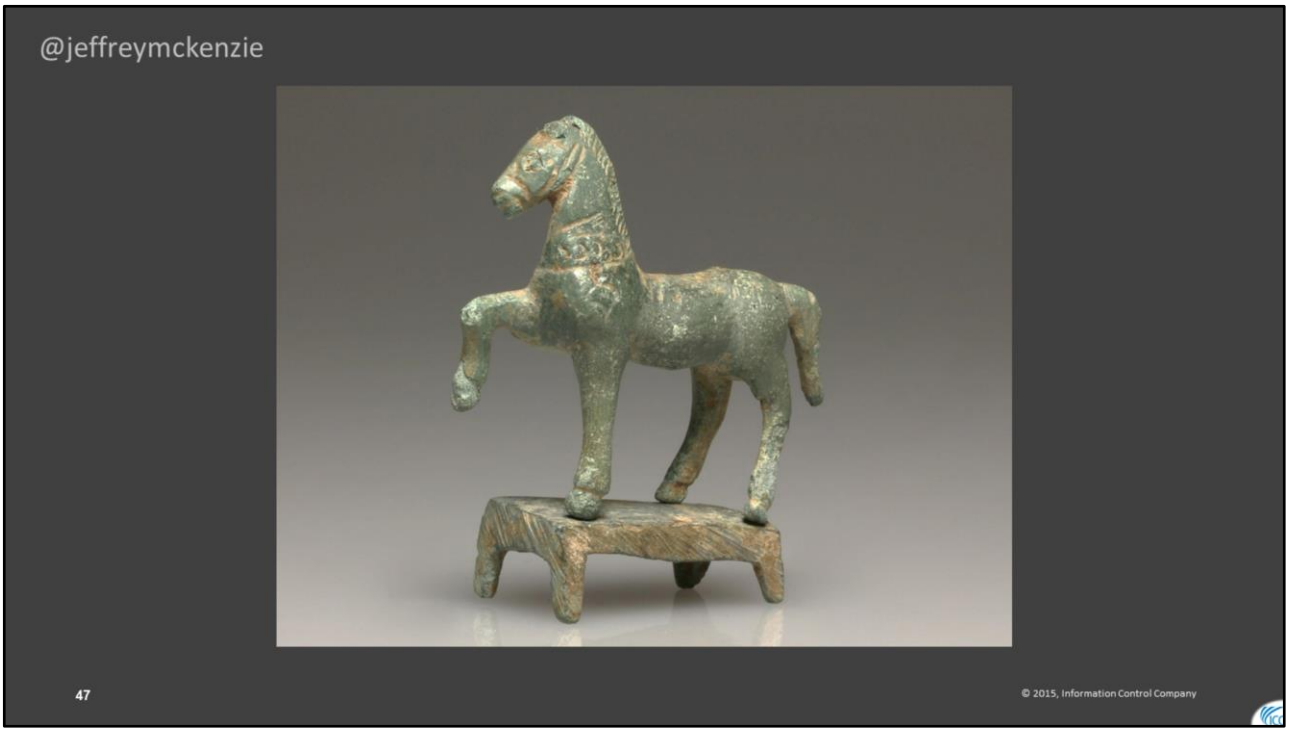
[image:]



Everything was going pretty well, they were back to their old production schedule.

After some time the rulers decided they wanted to experiment with different kinds of horses…..

[image:]



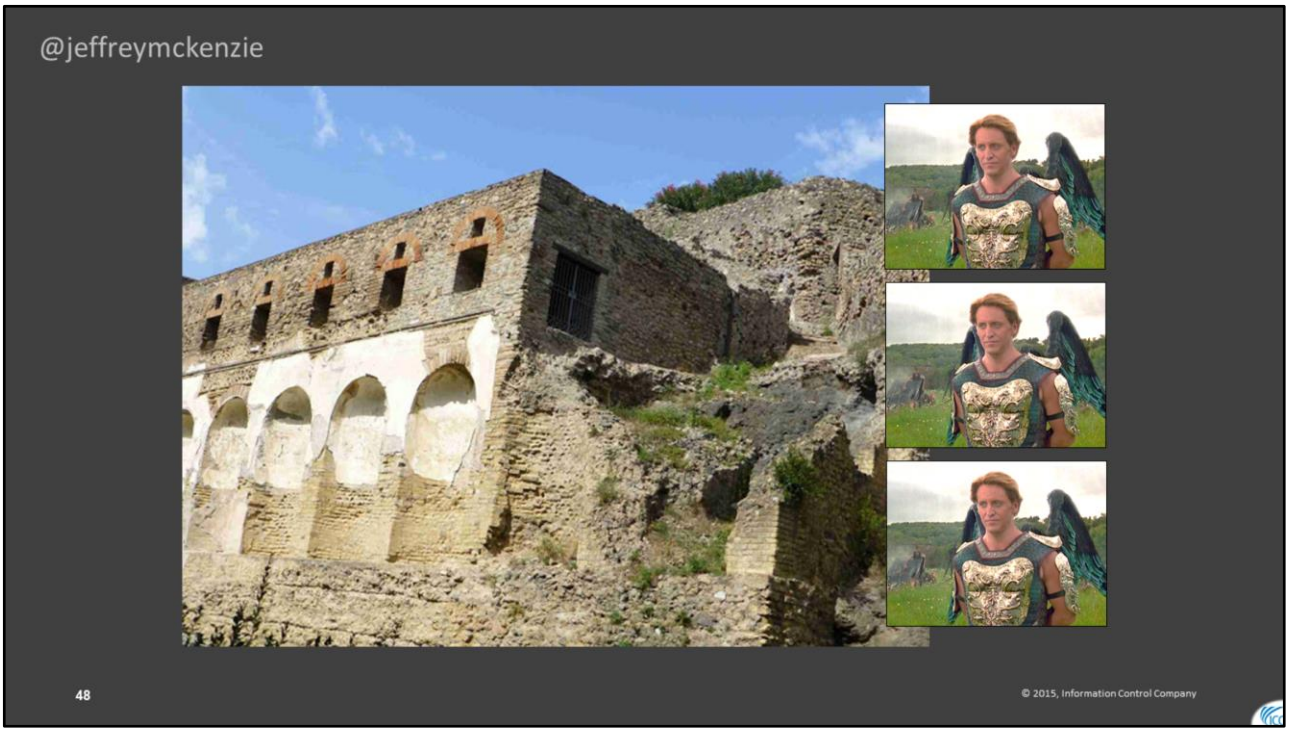
….in addition to the standard black wooden kind.

Max’s team was busy with the wooden horses,
so the rulers hired other teams to build the new kinds.

Whenever a new book was a created for a
different type of horse,

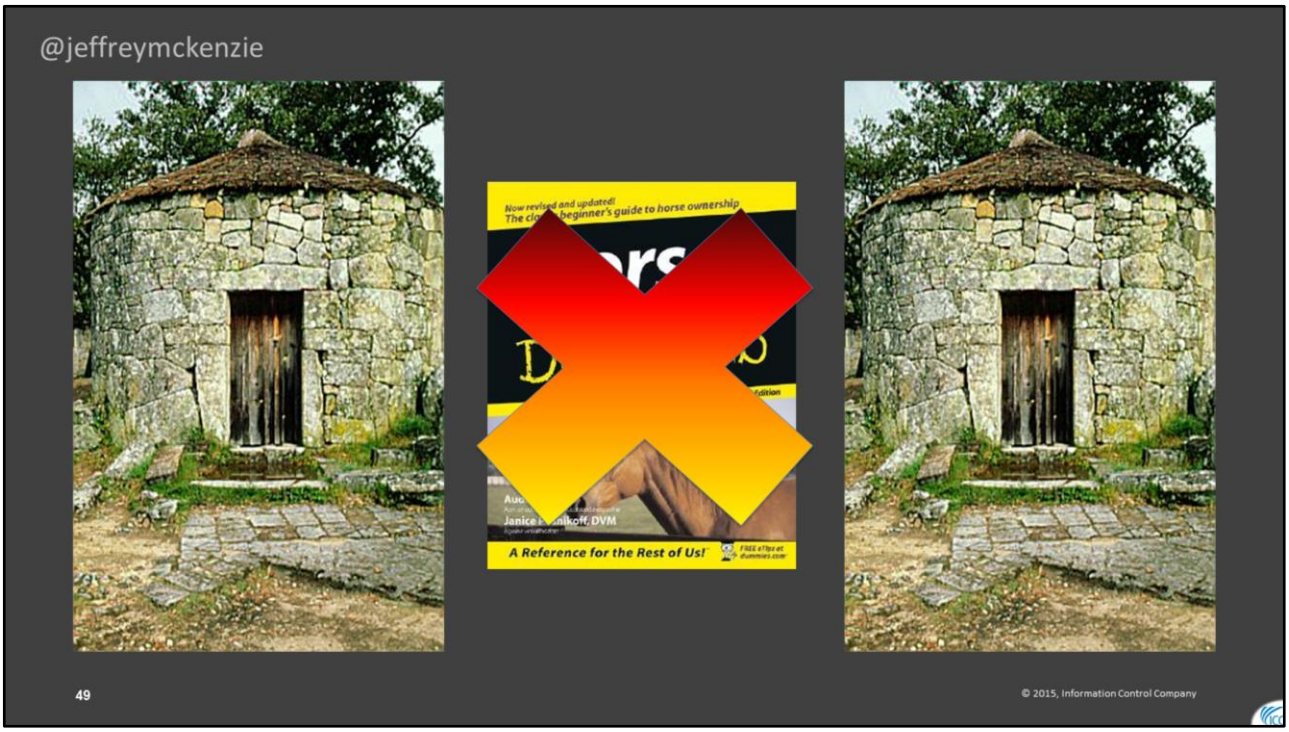
Another workshop was added specifically for
those workers who built the new type of horse.

[image:]



And each workshop had a dedicated messenger
who would be responsible for bringing changes
back and forth from the room to the central library.
And things started to get more complicated.

[image:]



If the group of workers needed to add
a longer tail to a horse,
they couldn't just hand the updated instructions
to the workers next door.

[image:]

Because the rulers wanted everything to
go through the central library council,
They workers would give the changes to the messenger,
who would take them to the central library –
then new books would be sent to
any room that needed one.

[image:]

@jeffreymckenzie

51

As the rulers pushed for more types of horses,
pretty soon the central library looked like this….

[image:]

@jeffreymckenzie

and the workers building like this.

[image:]

So the story of Centropolis illustrates some of the strengths and weaknesses of Centralized version control.

Centralized means that there is one, authoritative master copy of the source code, which is kept on a central server.

This provides a high level of oversight with regard to changes.

This type of version control is usually chosen for projects with a lot of components, where you have a large team of different skill levels in one physical location.

This setup would be ideal at a bank, for example, where you have a single enterprise product, and a lot of junior and senior developers under the same roof.

[image:]

This type of version control is usually chosen for projects with a lot of components, where you have a large team of different skill levels in one physical location.

This setup would be ideal at a bank, for example, where you have a single enterprise product, and a lot of junior and senior developers under the same roof.

[image:]

@jeffreymckenzie

55

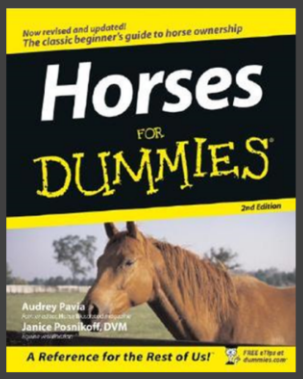
[CLICK]
In our story, the central library is essentially
the source code on the server,
[CLICK]
And the scribes are the version control system – the system
does the heavy lifting of copying and combining changes for you.
[CLICK]
The council of advisors would be code reviewers
and version control system administrators –
They decide what goes into the system and
where all the changes fit throughout the code.
[CLICK]

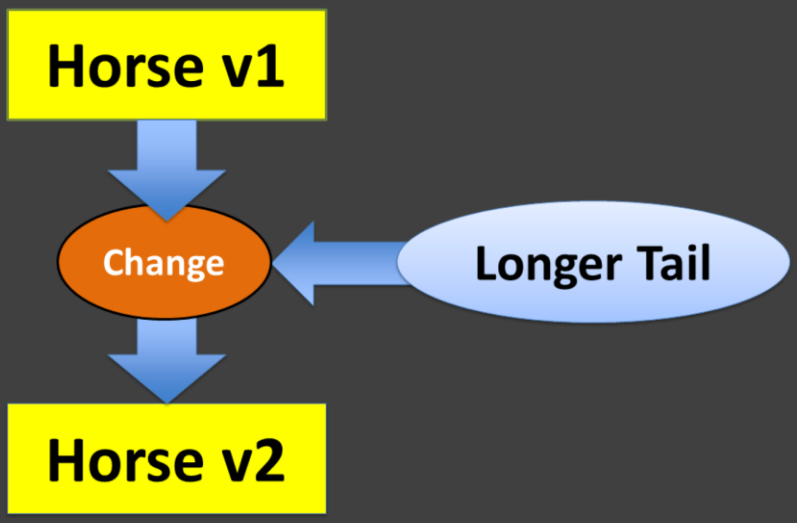
[image:]

And the various workrooms are roughly analogous to branches in version control. When you make a copy of existing code in order to make separate changes to it, that's called a branch (like creating a copy of the wooden horse instructions to make different colored horses)

[image:]

@jeffreymckenzie



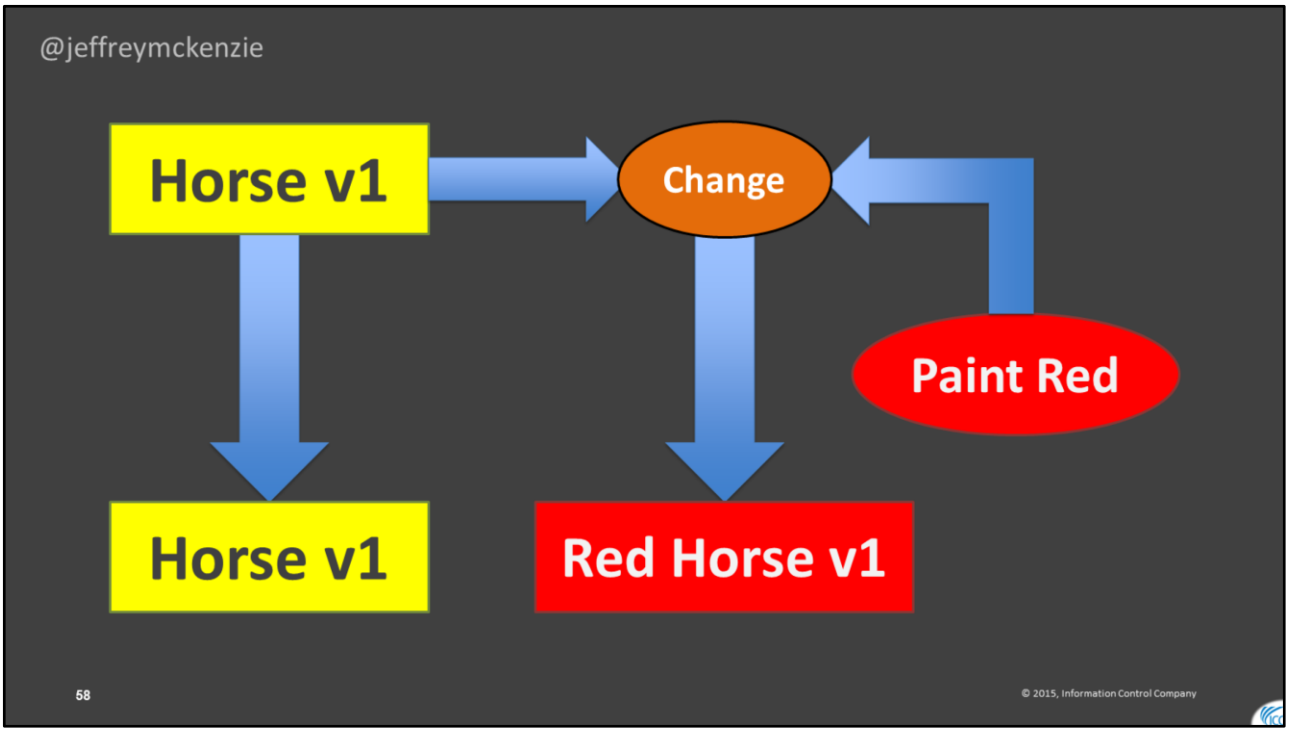


57

Centralized version control works the best when the changes are simple.

In this example, we have version 1, then a longer tail is added, and we have version 2.

[image:]

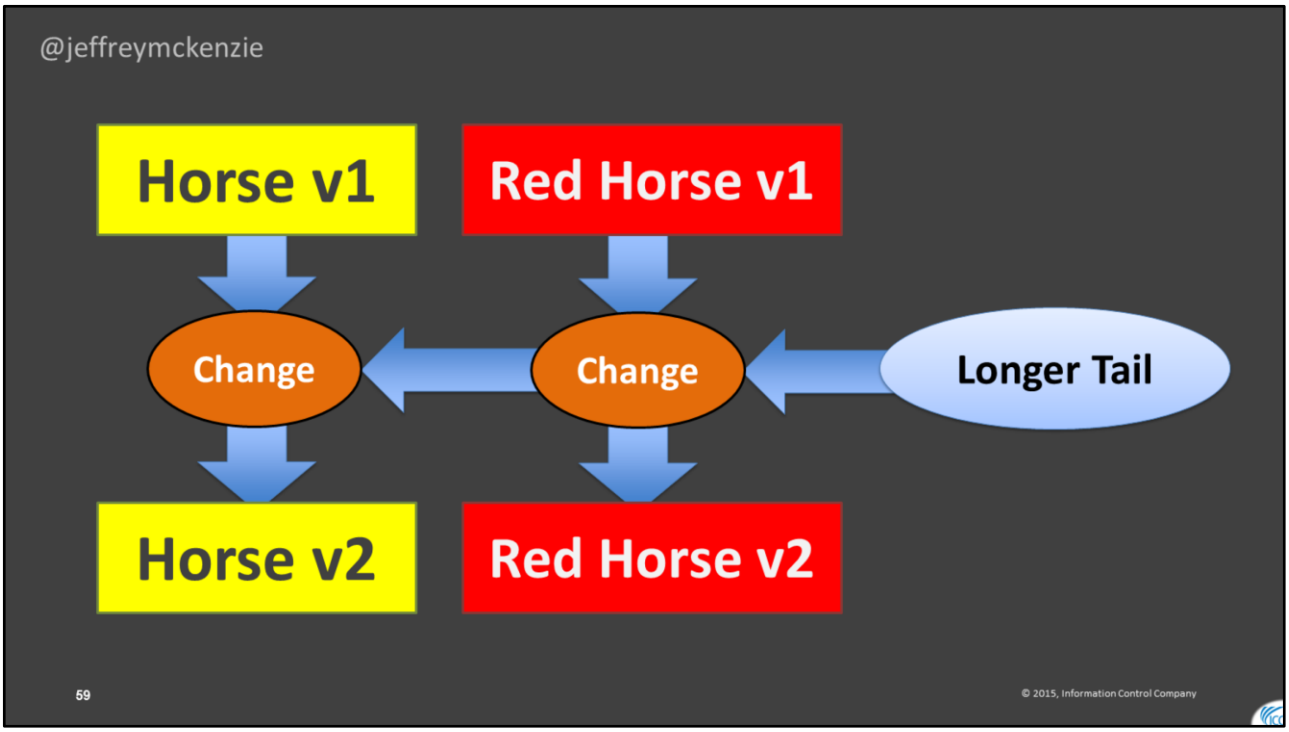


It gets more complicated when
we have to create new types.

For instance, if we create a different color,
we now have essentially two different
rooms to work in.

In this case, The red horse is a branch.

[image:]



Now if we want to make a change
that affects all horses,

we have to make the change in every branch.

A Centralized version control system
can help you do this, but it is still difficult
to plan and manage all these changes.

[image:]

@jeffreymckenzie

60

But this system was working well in Centropolis because of the all the procedures and processes that the rulers had put in place.

[image:]

By this time, Max and Reggie had been promoted several times,
and were now members of the council in the central library,
Reviewing changes and making decisions as to which changes
needed to be added to which types of horses.

They were making good money,
and business was as good as it had ever been….

[image:]

But what they really wanted
was just to make horses again.
So they got permission from the rulers
to be released from their service,
And started down the road
to the next part of our story….

[image:]

Gitlandia wasn't so much of a place as it was
a collection of cities, a kind of co-op.

First, Max and Reggie found a small town
to set up shop in, and started their own central library,
because they too wanted to make sure their instructions
were safe, and they liked how structured
And controlled everything was at Centropolis.

[image:]

@jeffreymckenzie

64

But because they were pretty proficient
in the entire process by now, they decided to
maintain everything in house.

Rather than have messengers to move
things back and forth, they moved the councils
and scribes to the workshop,

so that everything could be done faster.

[image:]

As they traveled between different cities
in the area, they ran into like-minded artists.

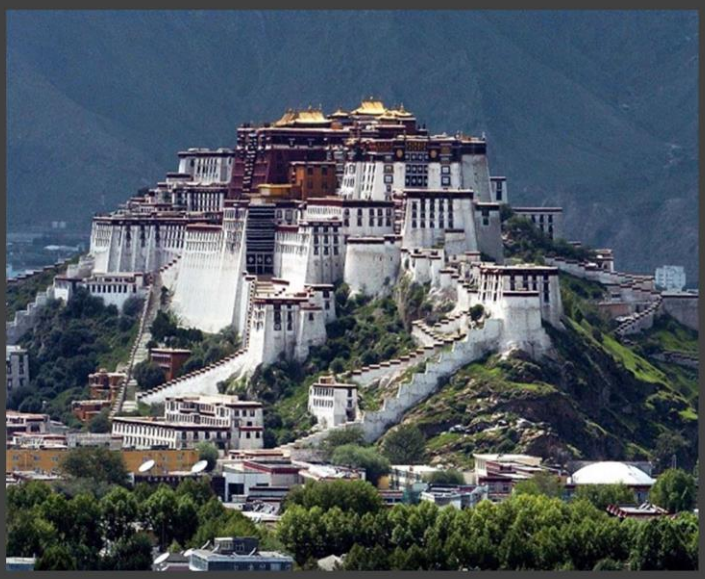
Rather than try to directly connect the cities
to a central library, they simply made a copy
of their library and gave it to the other artists.

That city would create its own council and scribes,
and use the same methods that Max and Reggie did.

Soon libraries were spreading across the region.

[image:]

@jeffreymckenzie



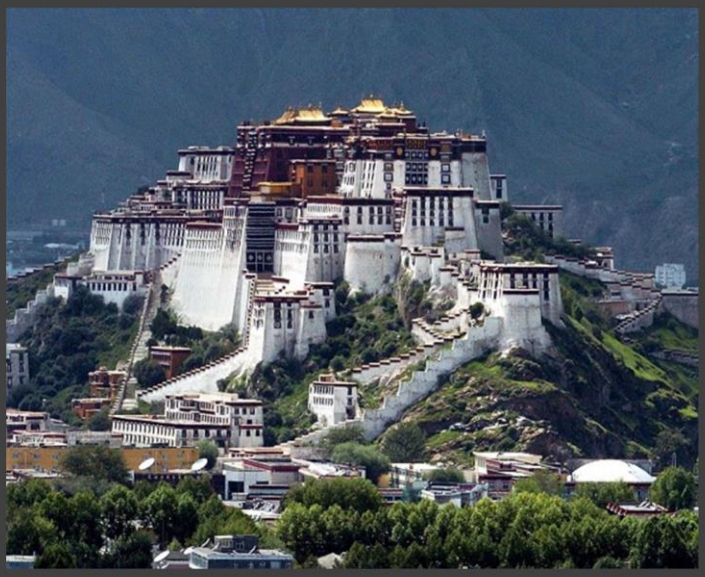
66

If two cities decided they wanted to work on
a new type of horse together, then those cities
would simply share their books and
changes with one another,

rather than having to make sure
everyone had all the changes.
system that so many people use today.

[image:]

@jeffreymckenzie

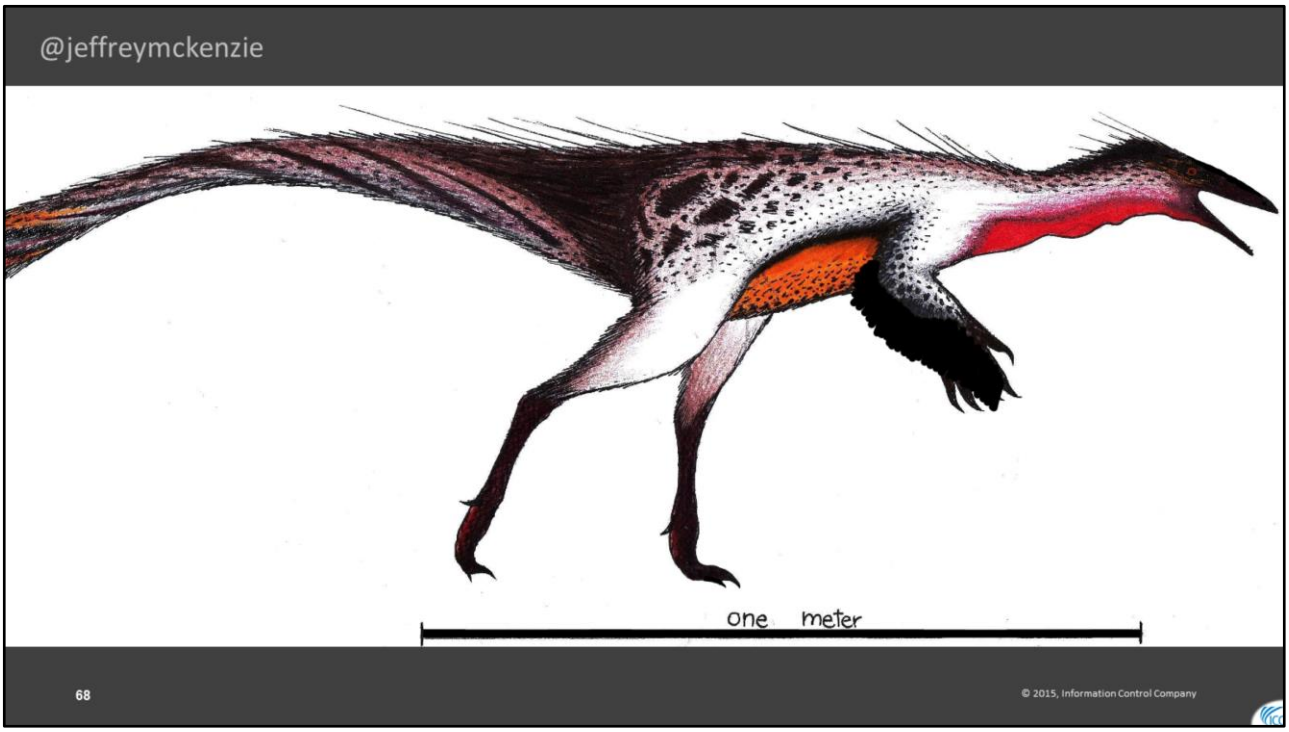


67

Eventually, as Gitlandia grew, all of its people decided that there should be one palace that would contain copies of all of the books from every city, but all cities were free to share books with one another.

They say that Gitlandia never really died out but much like the dinosaurs turned into birds….

[image:]



Gitlandia became the Git version control system…

[image: By Danny Cicchetti (Own work) [CC BY-SA 3.0 (http://creativecommons.org/licenses/by-sa/3.0)], via Wikimedia Commons]

that so many people use today.

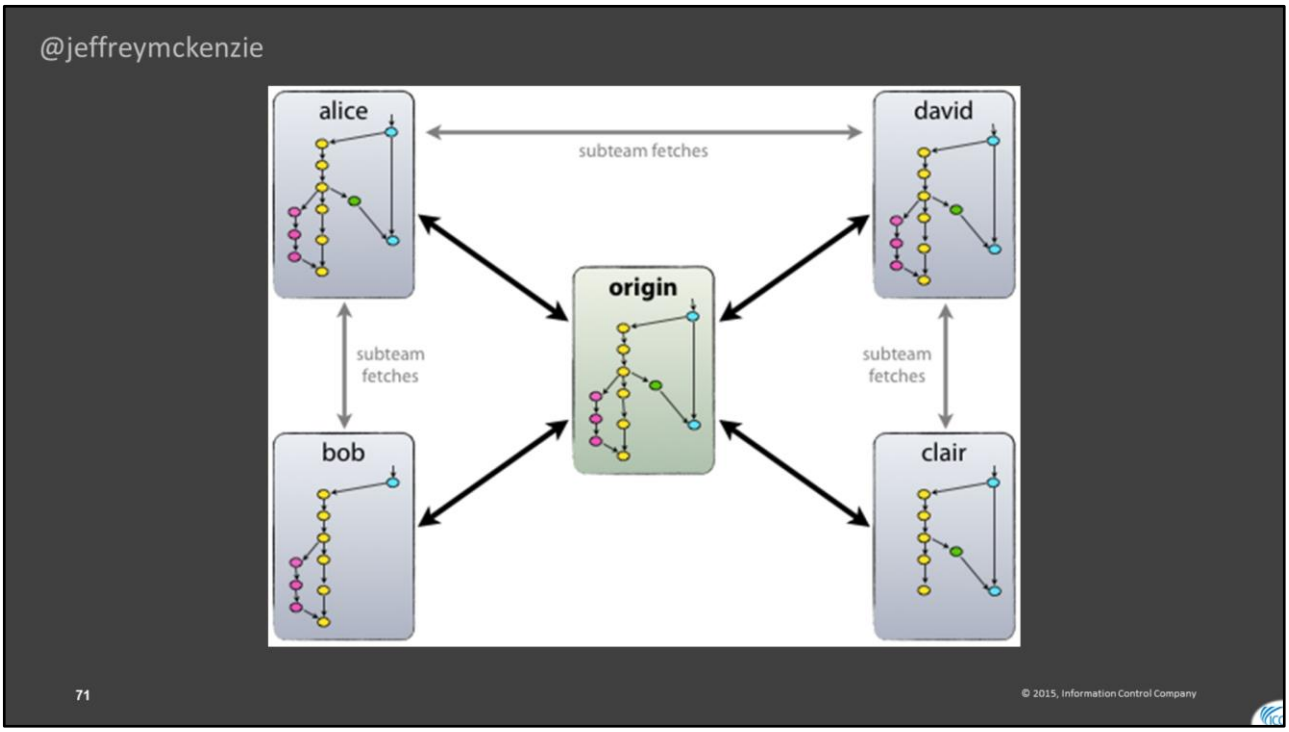
In Gitlandia, we have an example of distributed version control.

[image:]

Where centralized had the entire code
in one place, in distributed, a full copy
of the source code and version history exists
anywhere it's being worked on.

So there is no inherent, authoritative source –
By convention, however, there is usually
one version that is treated as the master copy,
and used for deployments and releases.
but there's not the same tight coupling
you have in centralized source control.

[image:]



This type of version control is often used with smaller, remote teams where the developer skill level is relatively high.

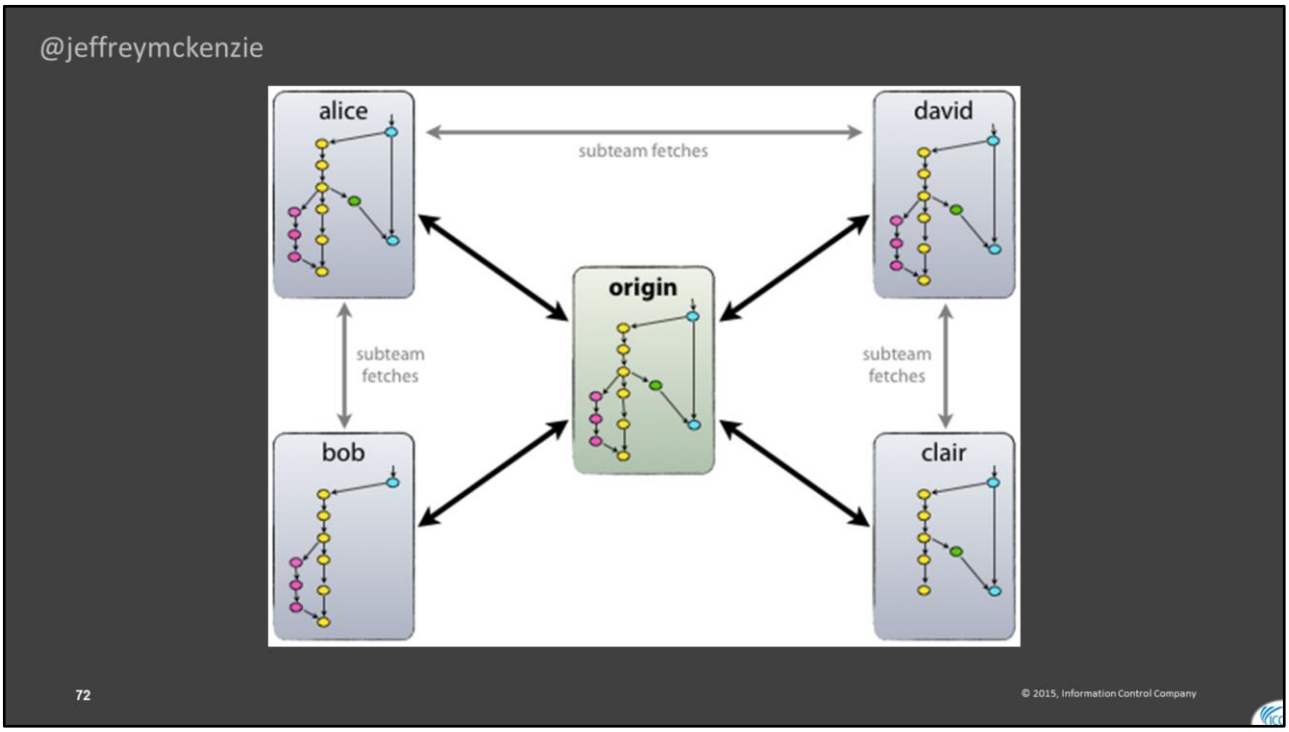
One of the benefits of working with the entire source control locally is that you can create local branches, and work on any number of features without cluttering up the main copy.

In Centropolis, any change that the workers made had to be sent to the central library.

In Gitlandia, the workers could experiment with their own changes locally, make local branches, commit whatever they wanted, and then when they had everything ready,

Shared their changes with everyone else.

[image:]



In Gitlandia, the workers could experiment
with their own changes locally, make local branches,
commit whatever they wanted, and then when
they had everything ready,

Shared their changes with everyone else.

[image:]

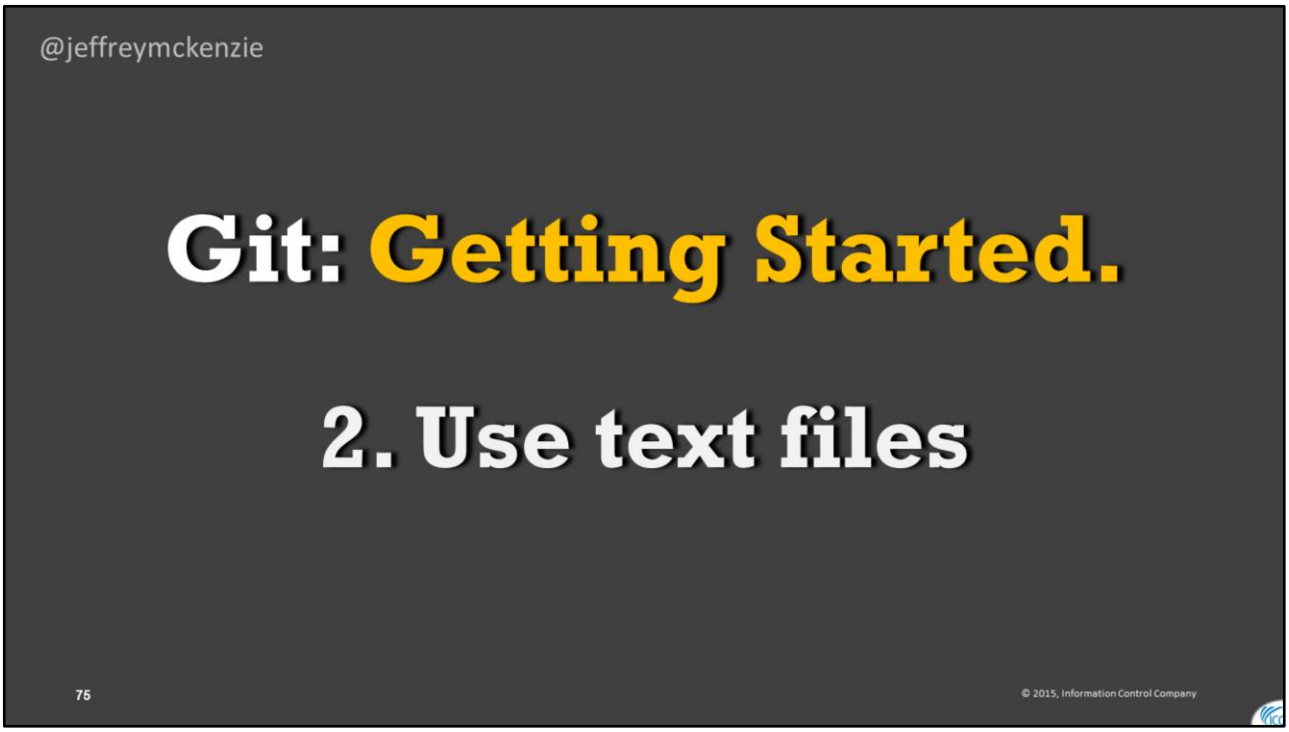
Now we are going to jump into Git.

You can download git from git- ESS CEE EM dot com
for the major operating systems, and there are instructions
for installation on how to get it up and running.

It's an fairly easy process to get going.

Here's a few of my recommendations
for getting started:

First, start small – don't try to use Git for the first time
on a significant project
(unless you have no choice)



Instead, try it on a simple text file and explore what you can do.

Or if you have a small throwaway project that you don't really need a full version control system on, try Git instead.

This overlaps a bit with the next point,
Which is to focus on the basics.

Master the simple workflow of
adding, modifying, and committing files.

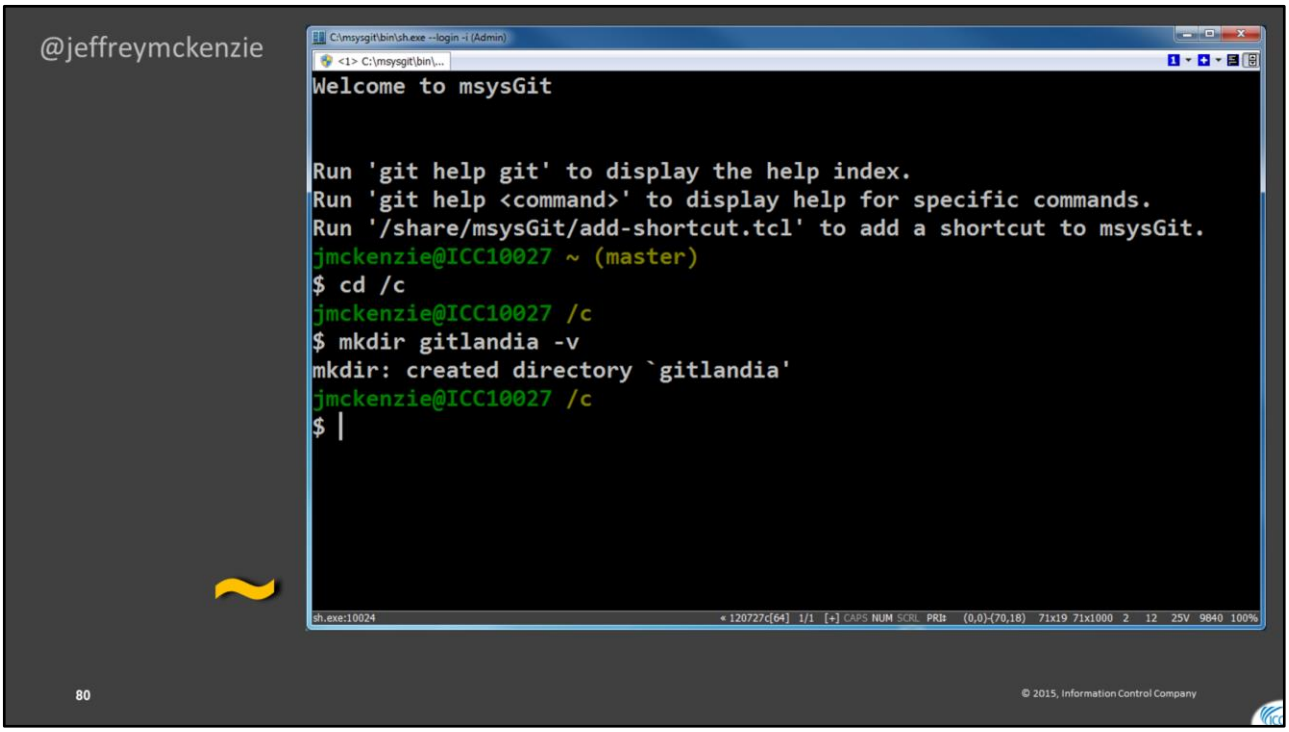
Git has a reputation of being hard to
learn and use, but its fundamental features
can be surprisingly easy, as we will see in a moment.

And finally, learn on the command line.
I know the command line strikes fear into
the hearts of even the bravest of us.
There are many graphical user interfaces
out there for Git, and eventually it probably
makes sense to use one of those at some point….

But what a graphical interface does is abstract away a lot of the lower level stuff. Using the command line allows you to experience exactly what Git is doing, and I think that helps you understand Git from a conceptual point of view.

So what I'd like to do is walk through the process
of the basic git workflow, looking at some of
the main actions and commands that are available.

This is not intended to be a full git tutorial,
but I wanted to go quickly
through the high-level steps
for those who haven't used it before.

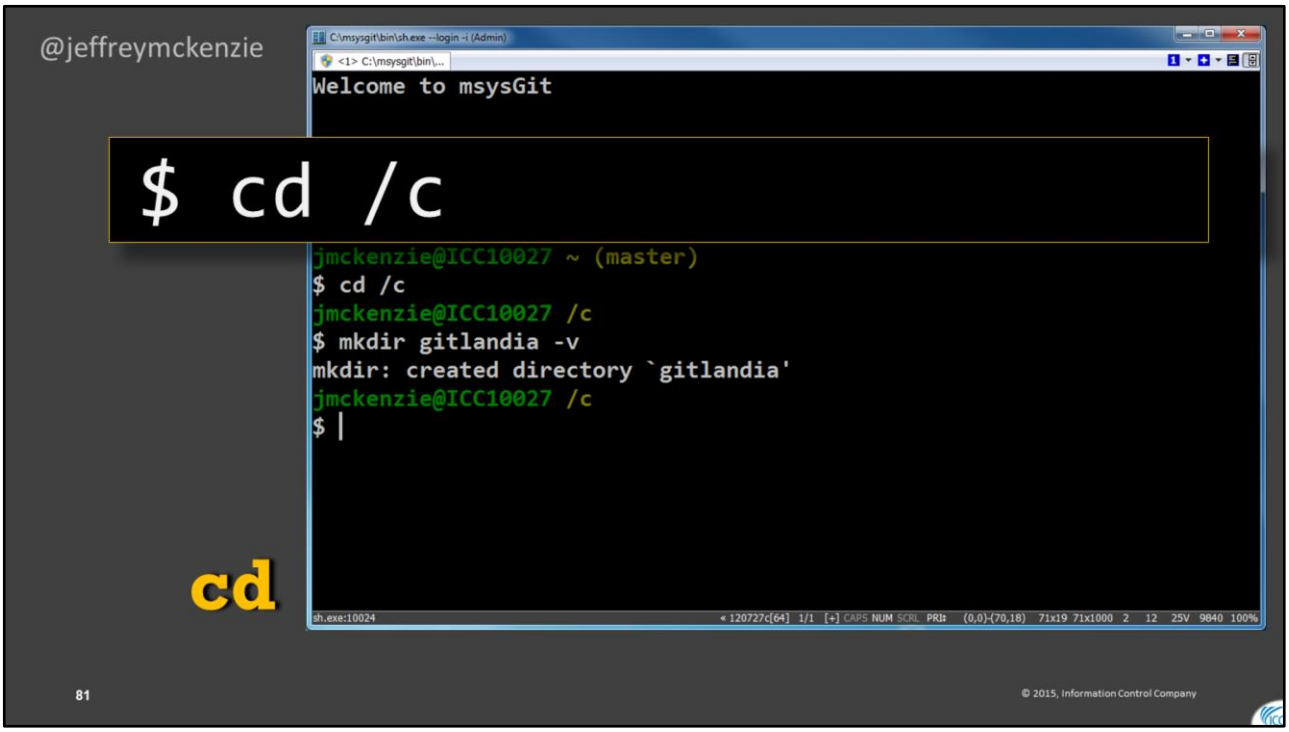


I'm using Git for windows, or msysgit, which comes with a bash shell that's similar to a Linux.

The dollar signs here are the command line prompts, where I can enter commands.

The console starts up in my home directory, so the first thing do here…..

[image:]

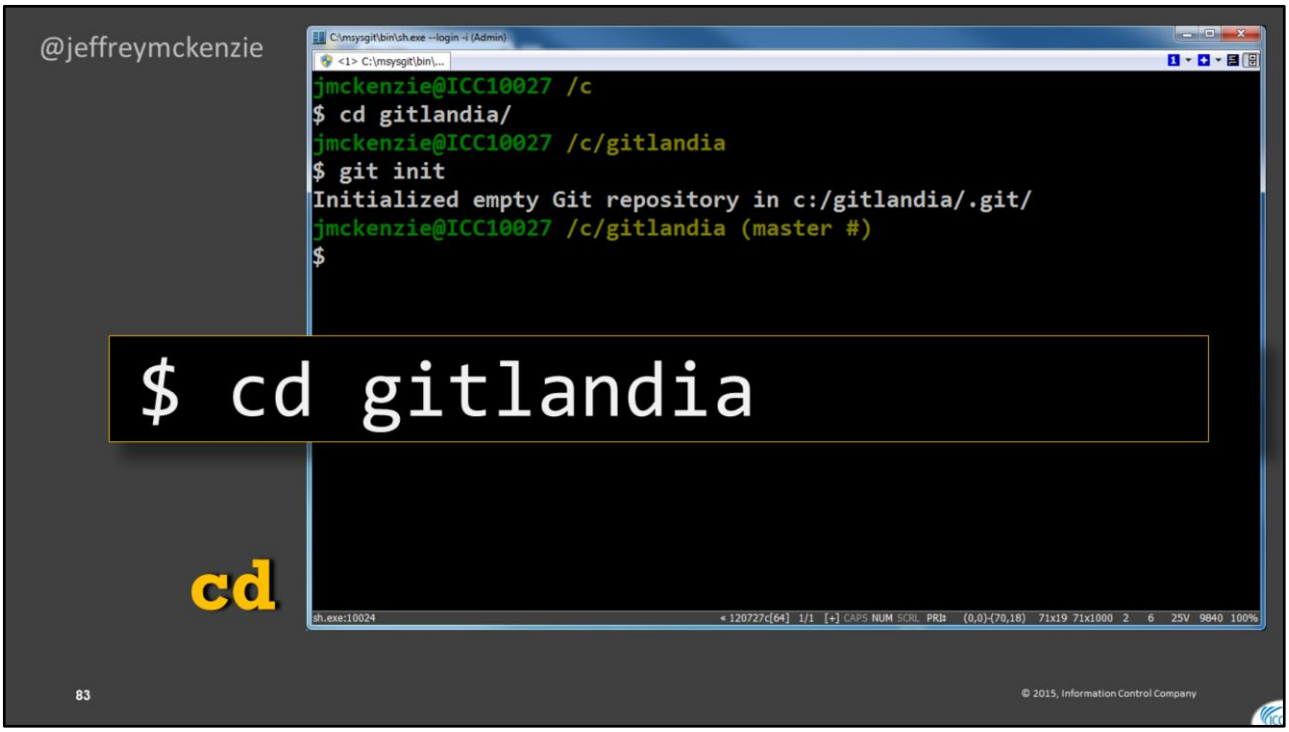


Is to CD, or change directory
to the root of the c drive.

[image:]

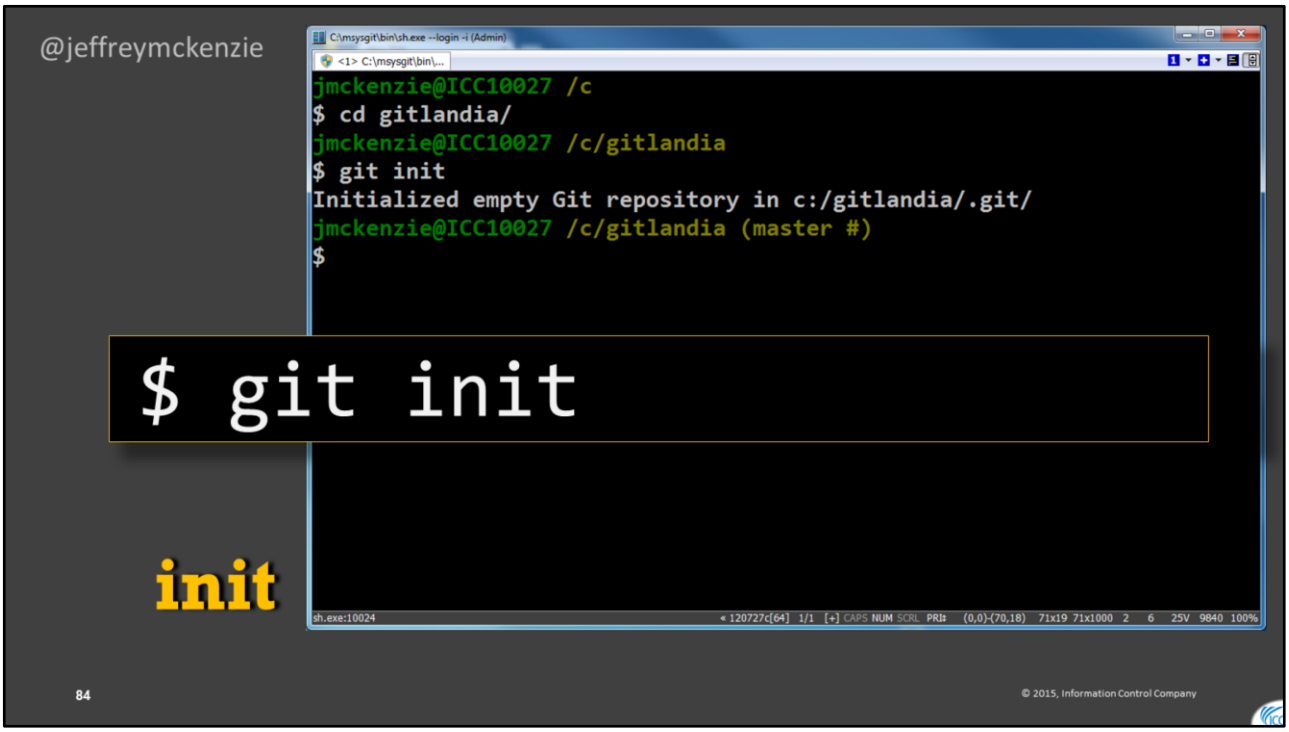
Then I create a gitlandia directory.
that's where my git repository is going to be.

[image:]



Next I change into the gitlandia directory….

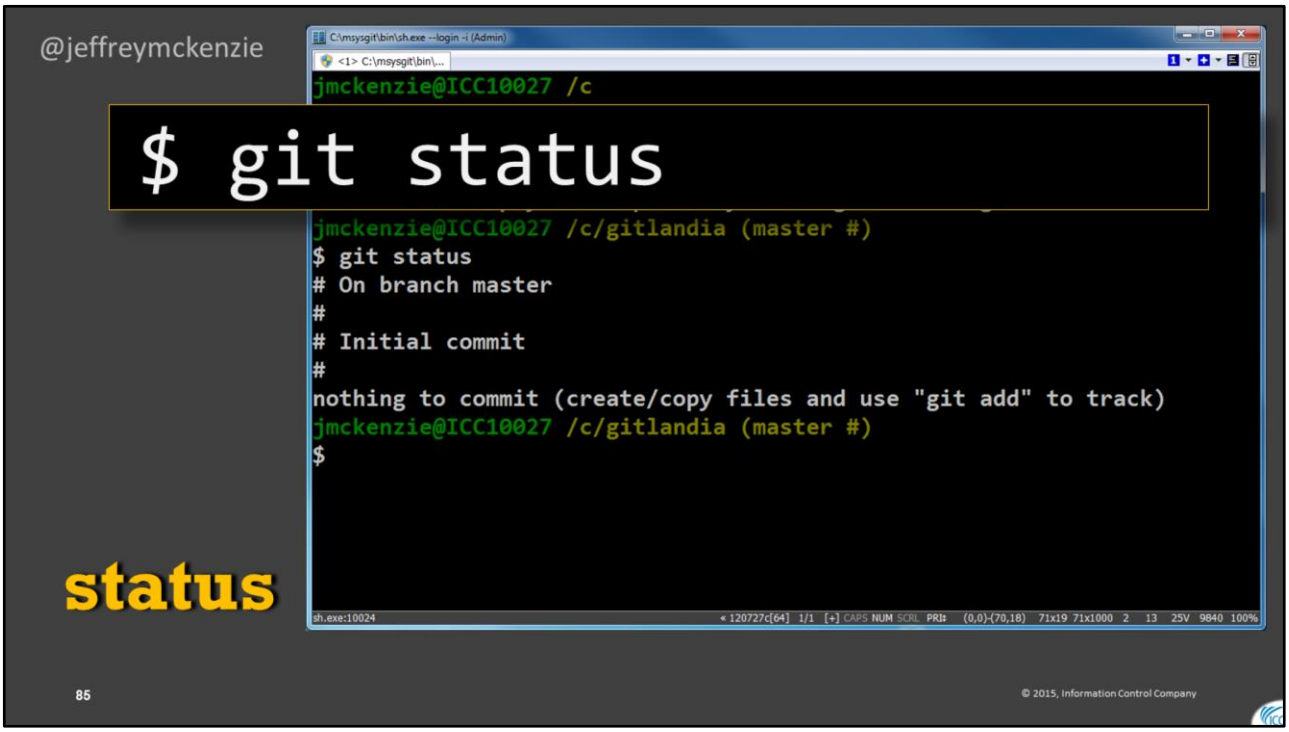
[image:]



Next I change into the gitlandia directory,
and run the Git Init command –
That command instructs git to create
everything needed to start using a git project.

By default it creates one branch called “master.”
This is similar to trunk in SVN

[image:]

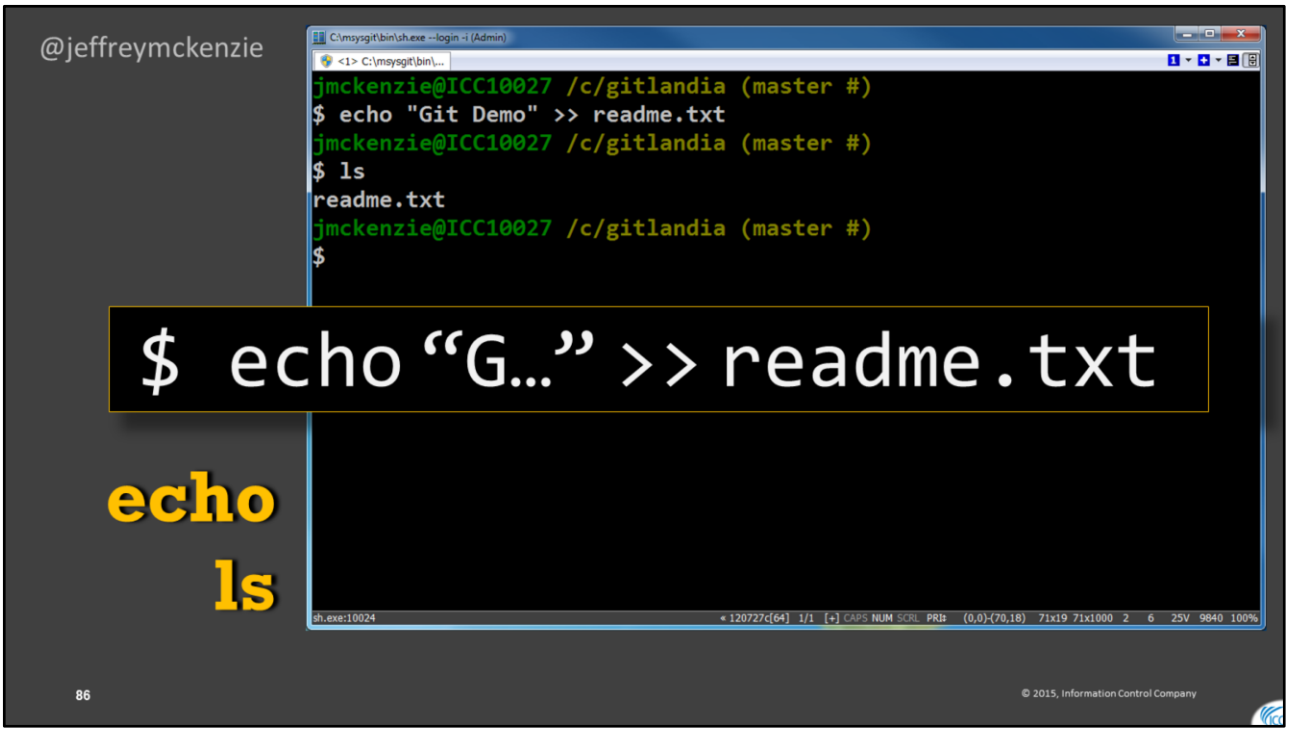


After that I can use the git status command,
which shows you the current state of your source code.

Because I just created it, it shows nothing to commit –
there are no files in the directory.

So we will create one.

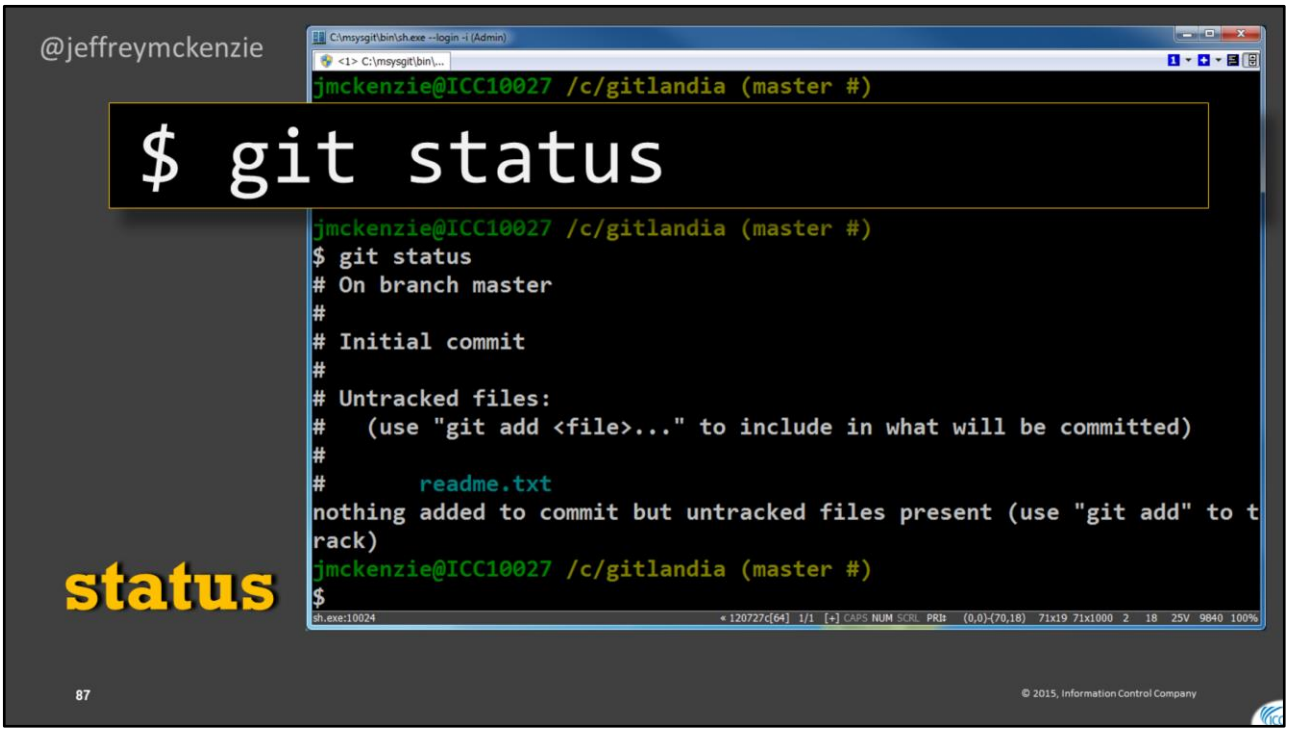
[image:]



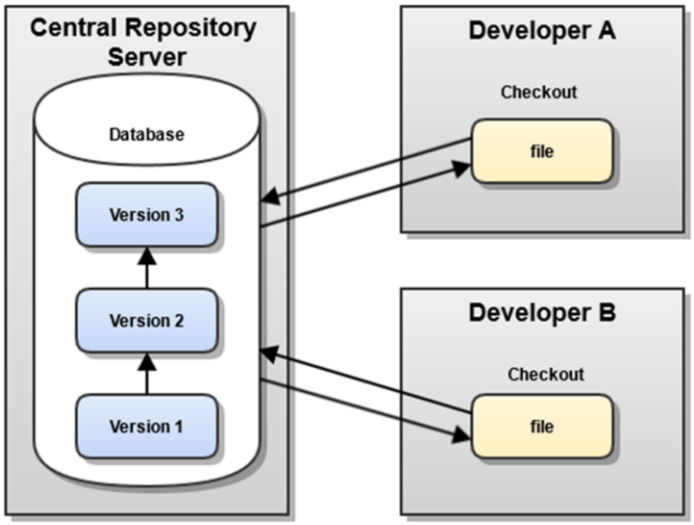
Here I'm going to create a simple text file
that I can add to git.

I'm using the echo command to
redirect text into a new file called readme.txt.

Then I enter the "LS" command to give
me the list of files, to verify that the file was created.



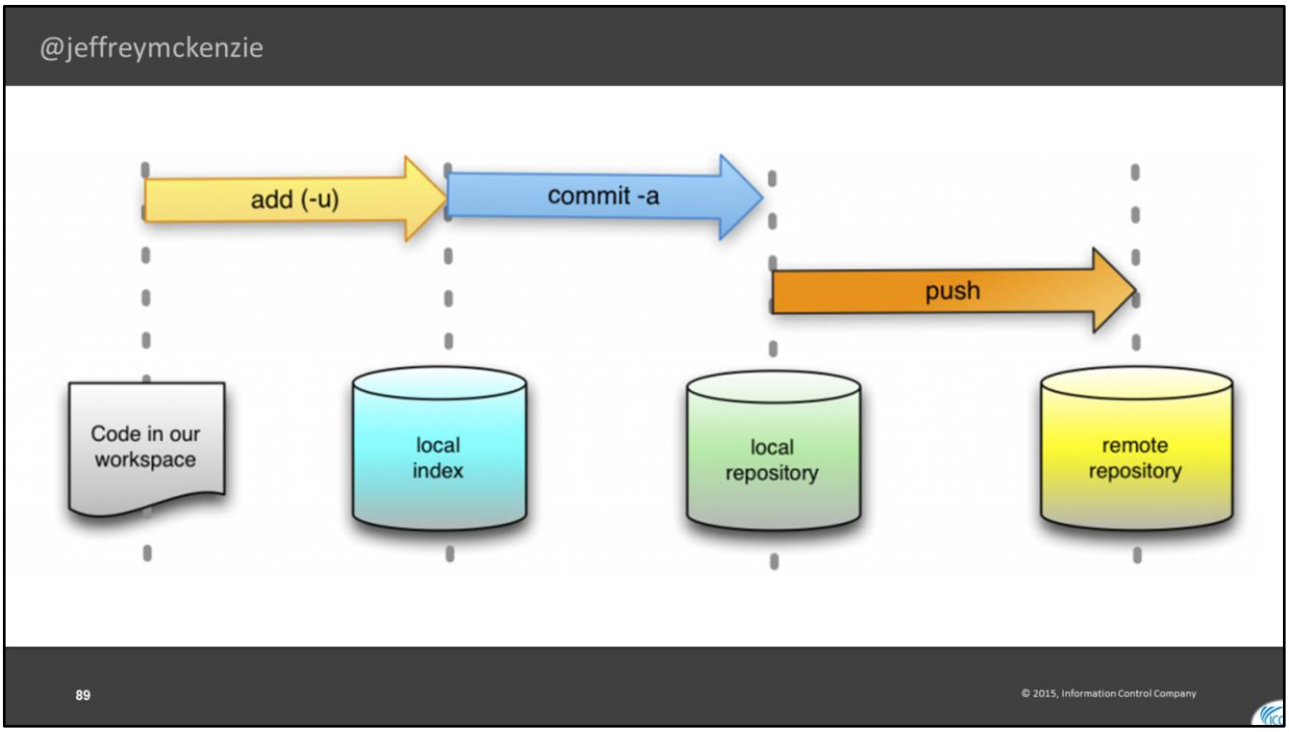
When I run the Git Status command again,
I get a different message.
It says “nothing added to commit
but untracked files present.”
So it’s telling me,
hey there’s a file here,
but I’m not keeping track of its versions yet.



Before we actually add the file, I'd like
to explain the difference between how files and changes
are added to git versus a centralized version control system.

The centralized model has pretty much two states:
Either the file is being edited locally, or
it is committed to the central server. That's it.
Either the developer is working on the file,
or the work is finished and saved on the server.

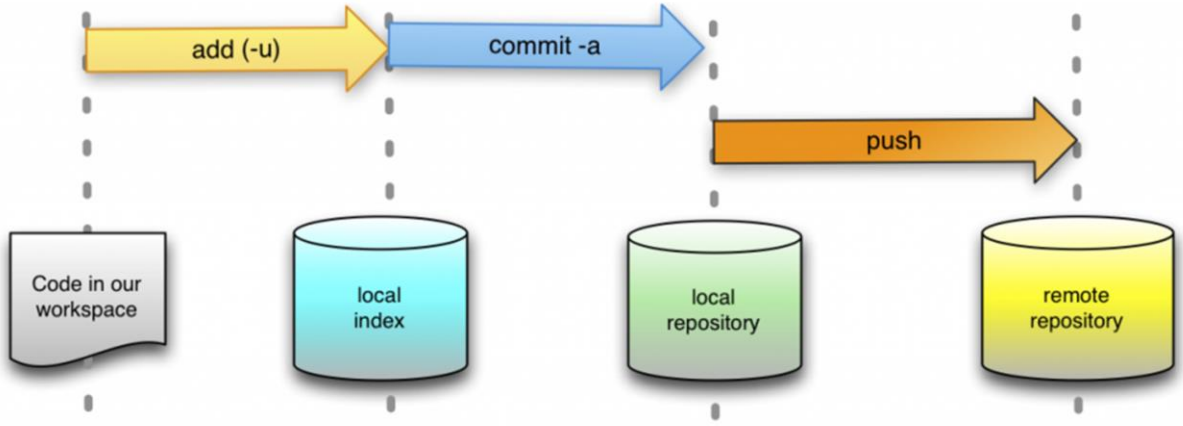
The git model is broader than that.



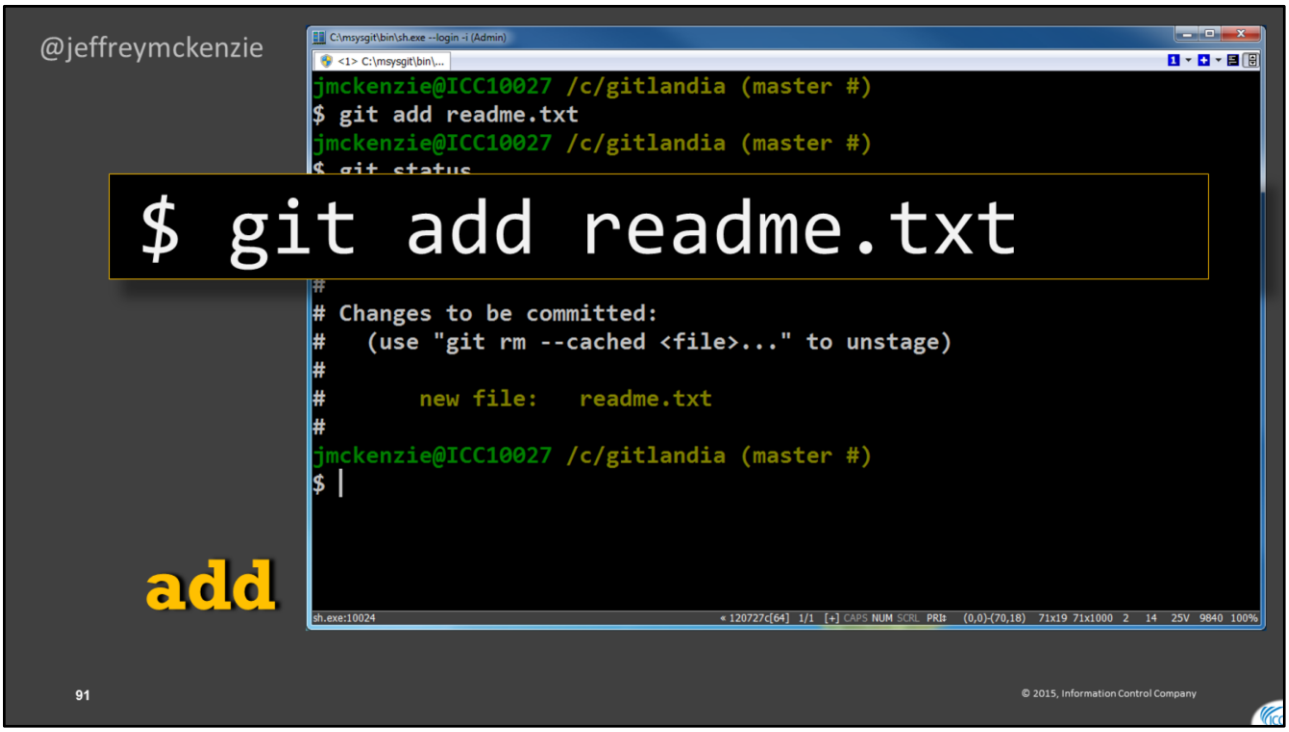
It has 4 states – first, the file can be in the process of editing,

Second, can be added to the local index (which is kind of a staging area that allows you to prep files for committing, before you actually commit them.

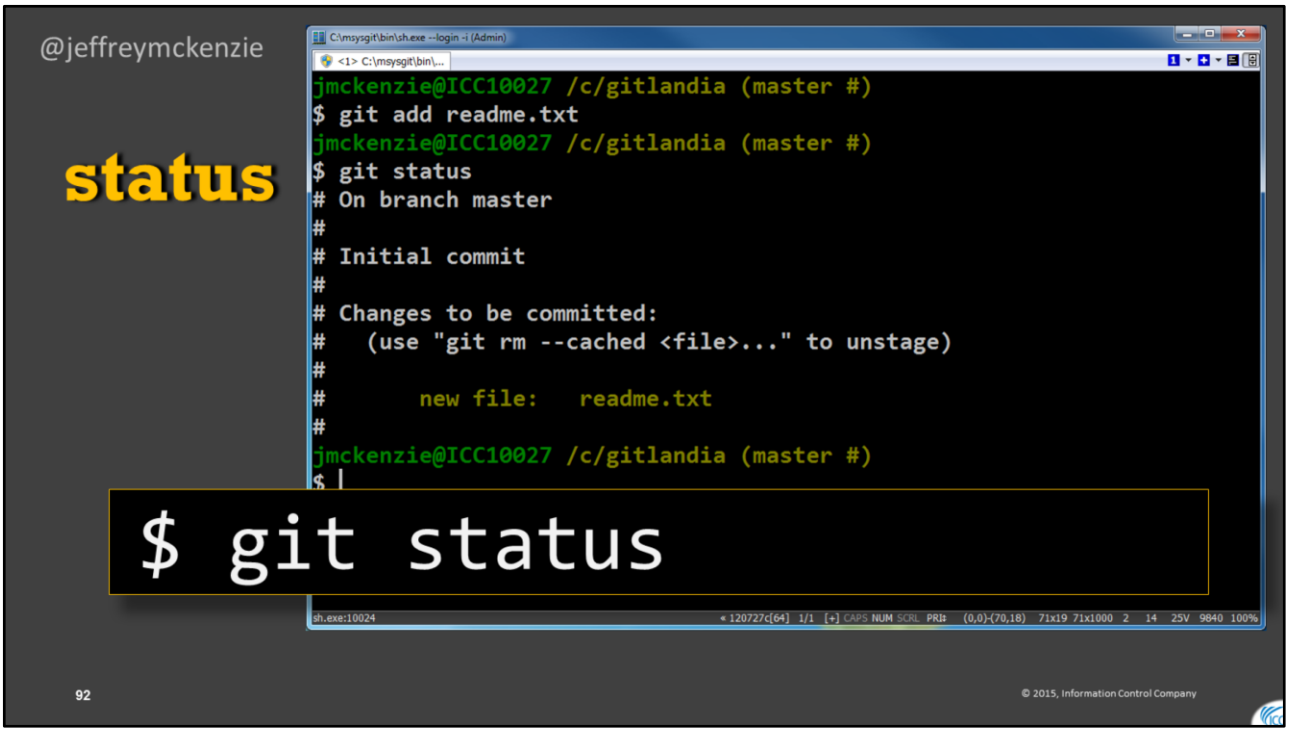
Third, It can be committed locally.



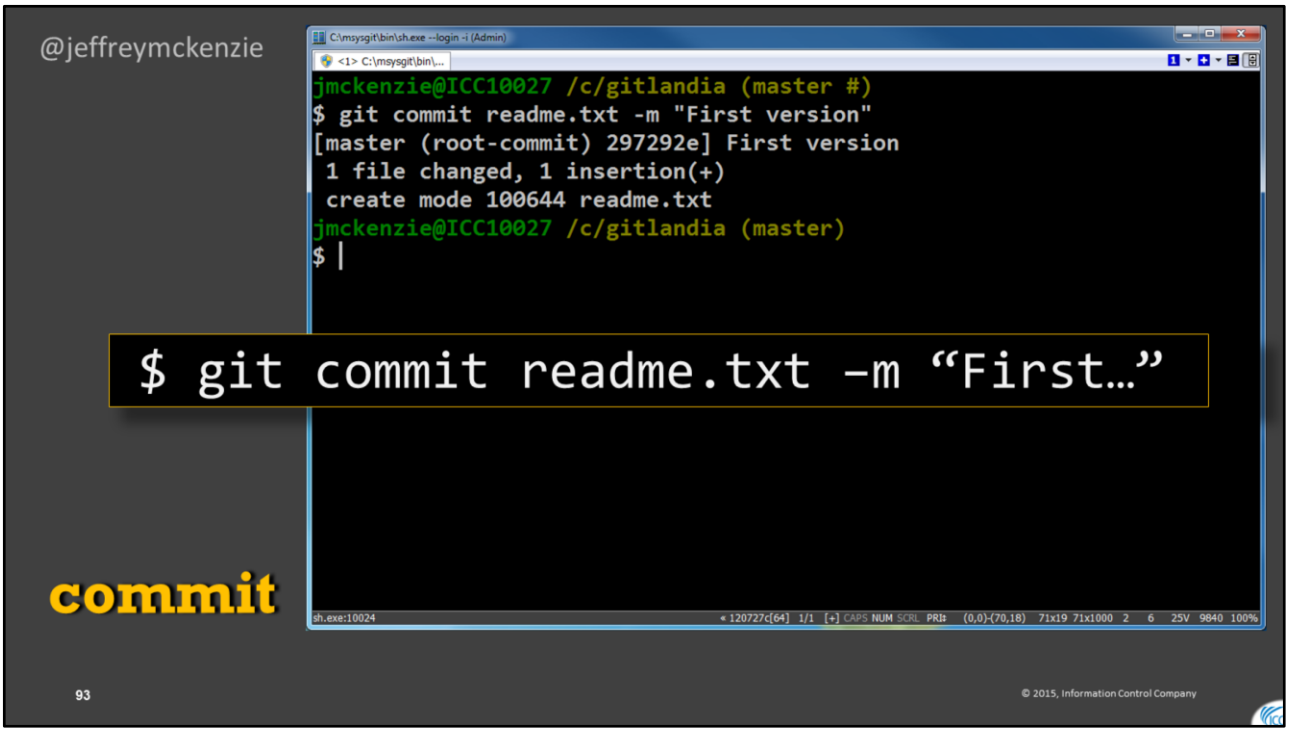
And 4th, it can be pushed to the remote server.
So Git adds two states in the middle
of local edit and remote commit.
This gives you a lot of flexibility to have
your own local history that's completely
separate from the server history,
And still maintain a relationship
to the master code.



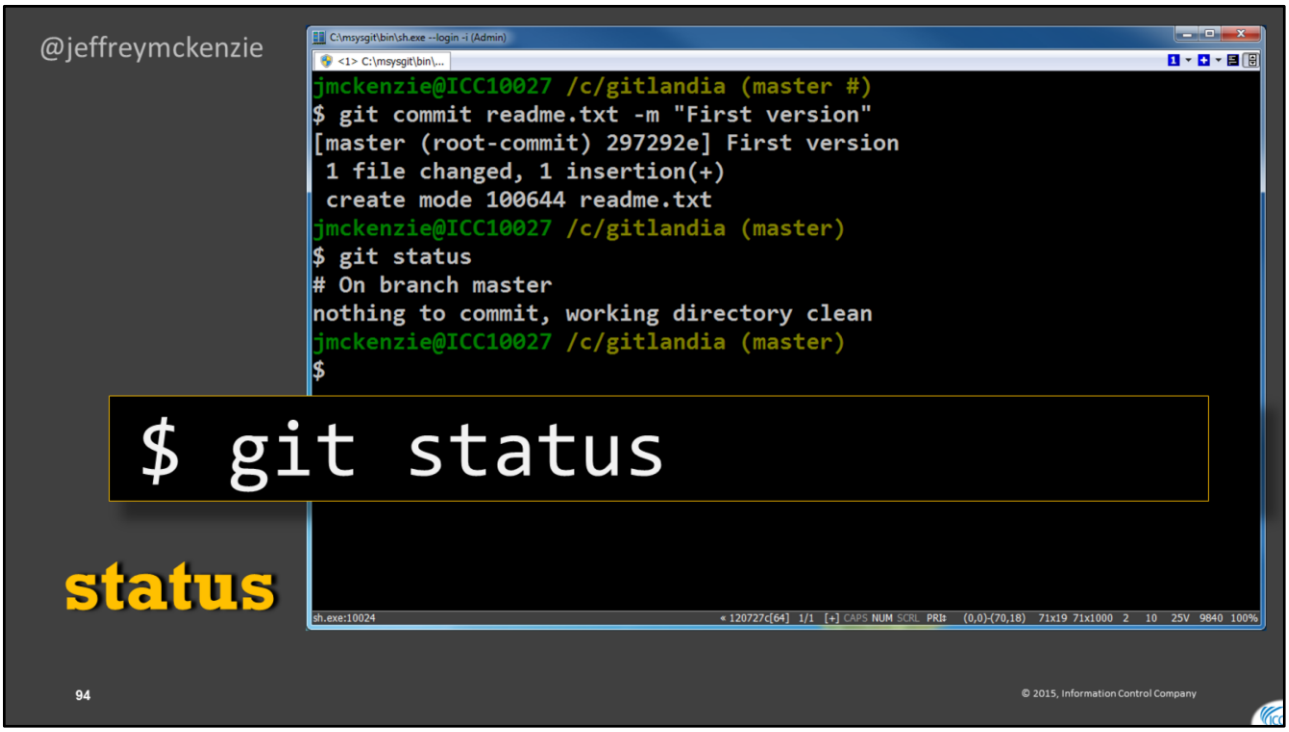
So the command to add a file
to that staging area or index is,
not surprisingly, Git Add.
So I add the readme.txt to Git.



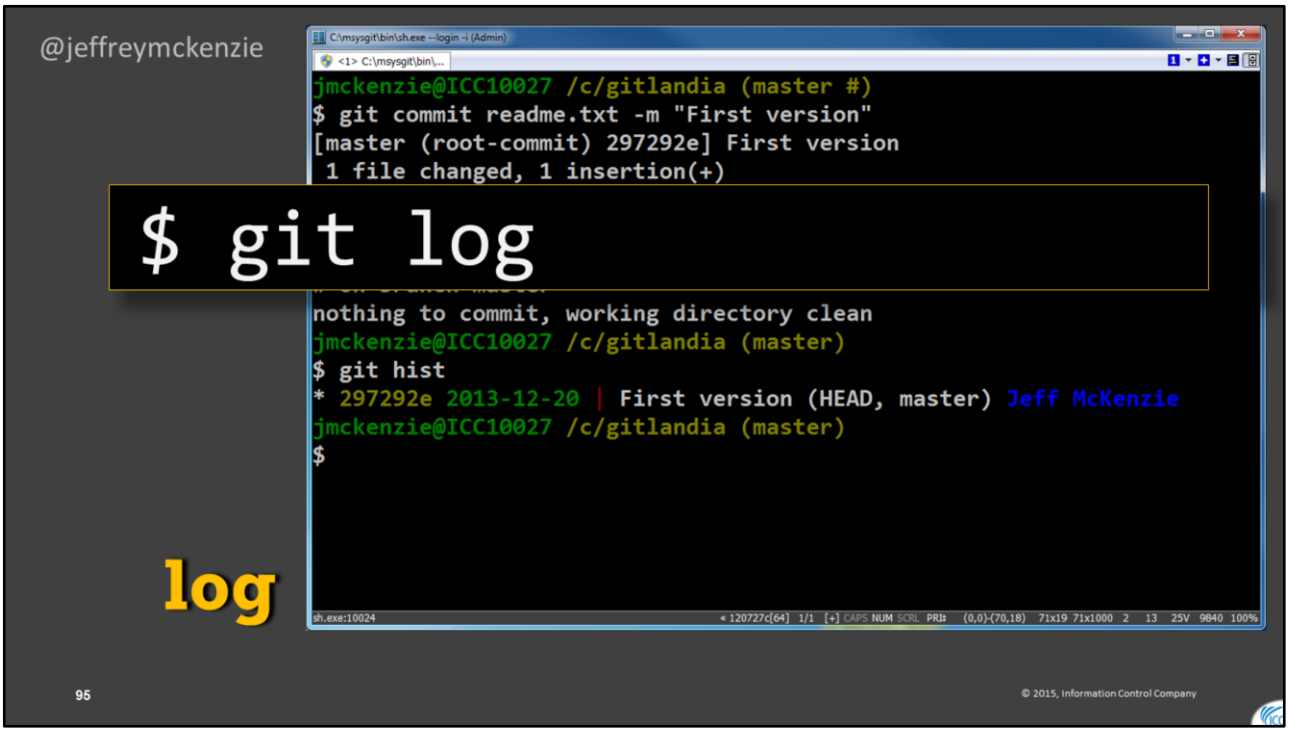
When I run git status again, it shows me that the readme file has been added to the local index.
It hasn't been committed yet, but it's ready to. So I can still change my mind and decide to remove it from the index.
It's not part of the history right now.



Now I can run the commit command
on the readme text file –
the “M” option indicates
that you are adding a message
to the commit.
A commit is the same thing as a
check-in on other version control systems.



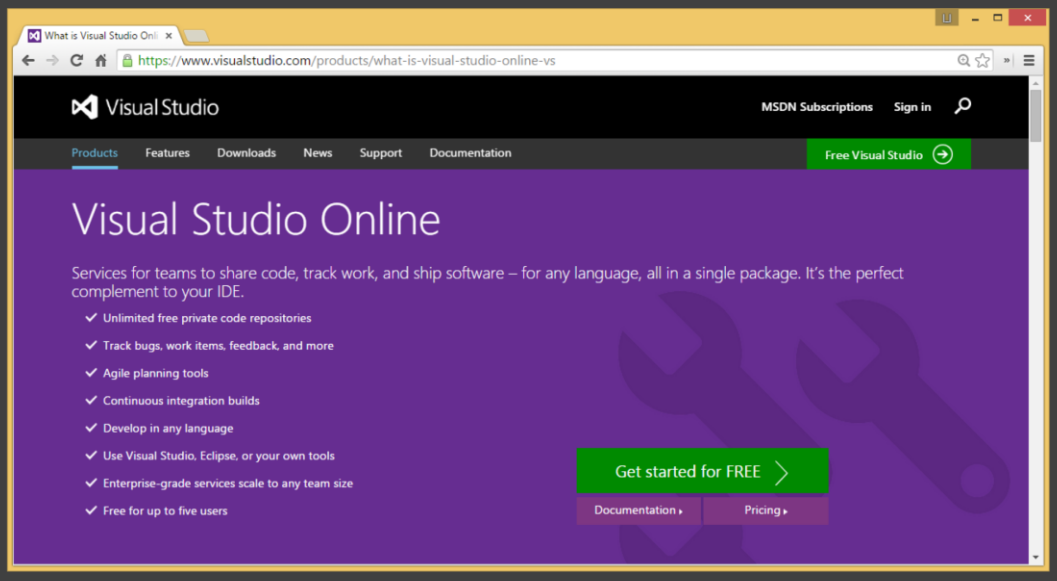
I run status again, and it shows that there are no outstanding changes – everything is checked in and clean.



There's also a command called Git log, which displays the revision history for the repository.

Here we can see the commit or checkin I just did, together with the date, my name, and the message I entered.

@jeffreymckenzie

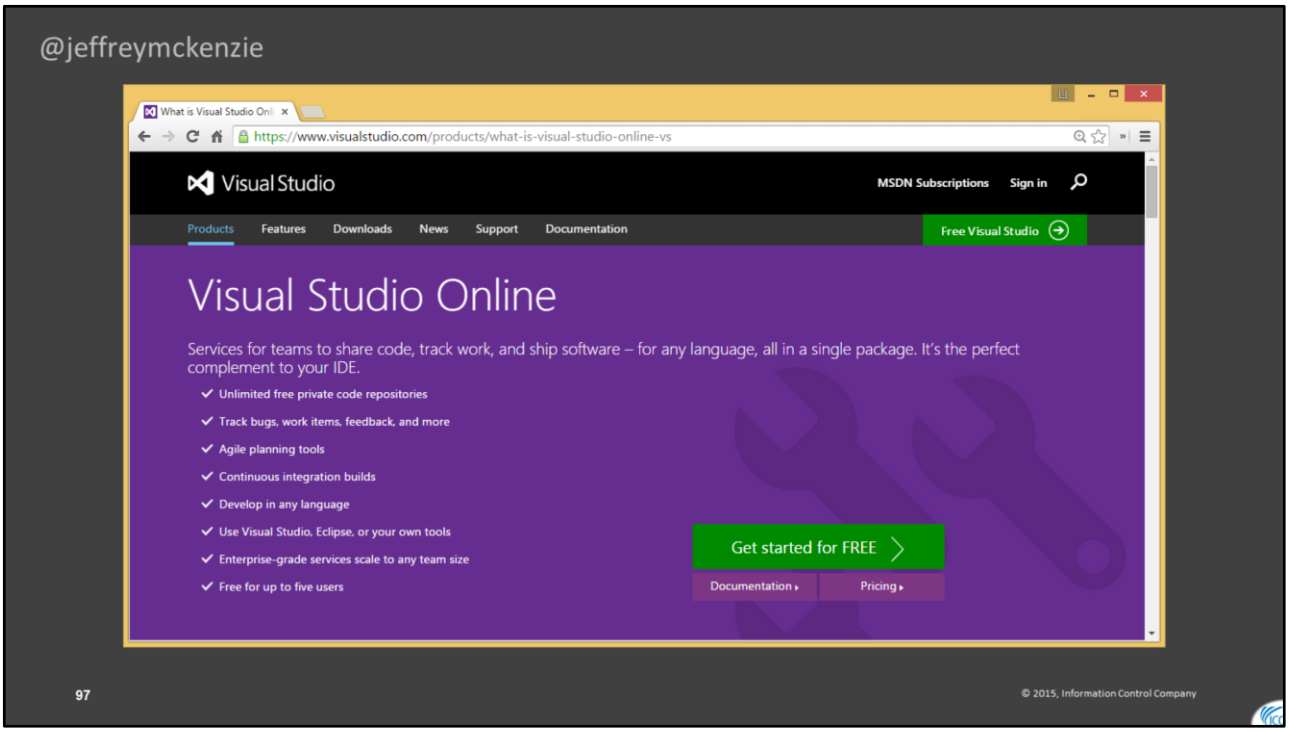


96

So let's take a look at some git hosting services, starting with visual Studio Online.

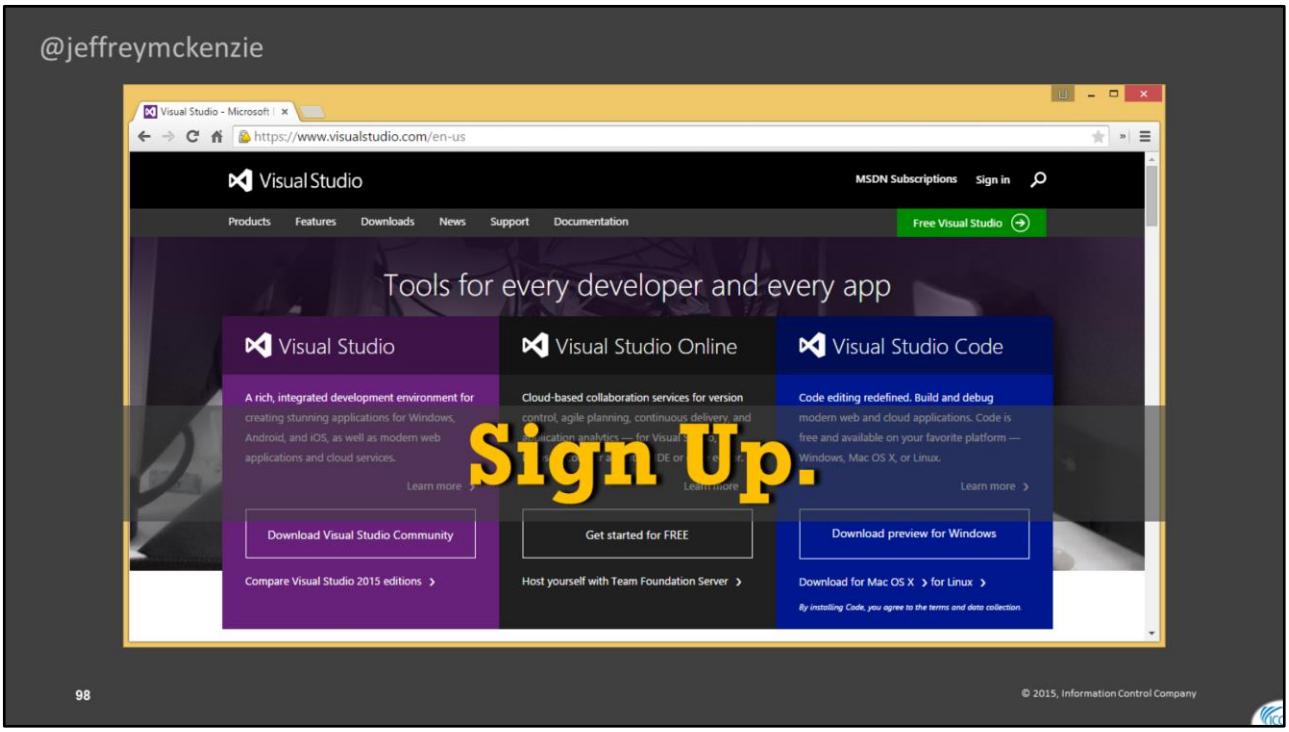
What we are going to do here is create a project, Then move or push our local git repository up to that project.

[anybody used VSO?]

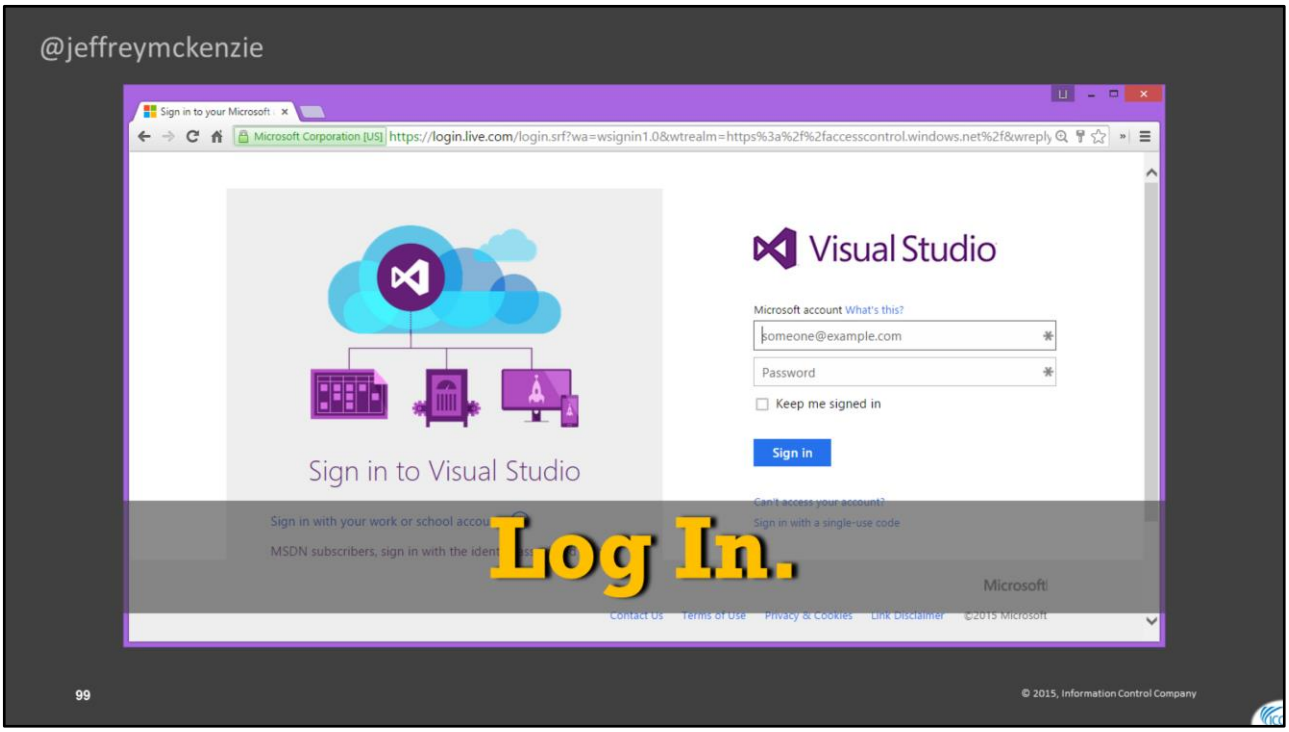


Visual Studio Online used to be called
Team Foundation Service,
and it is essentially TFS in the cloud.

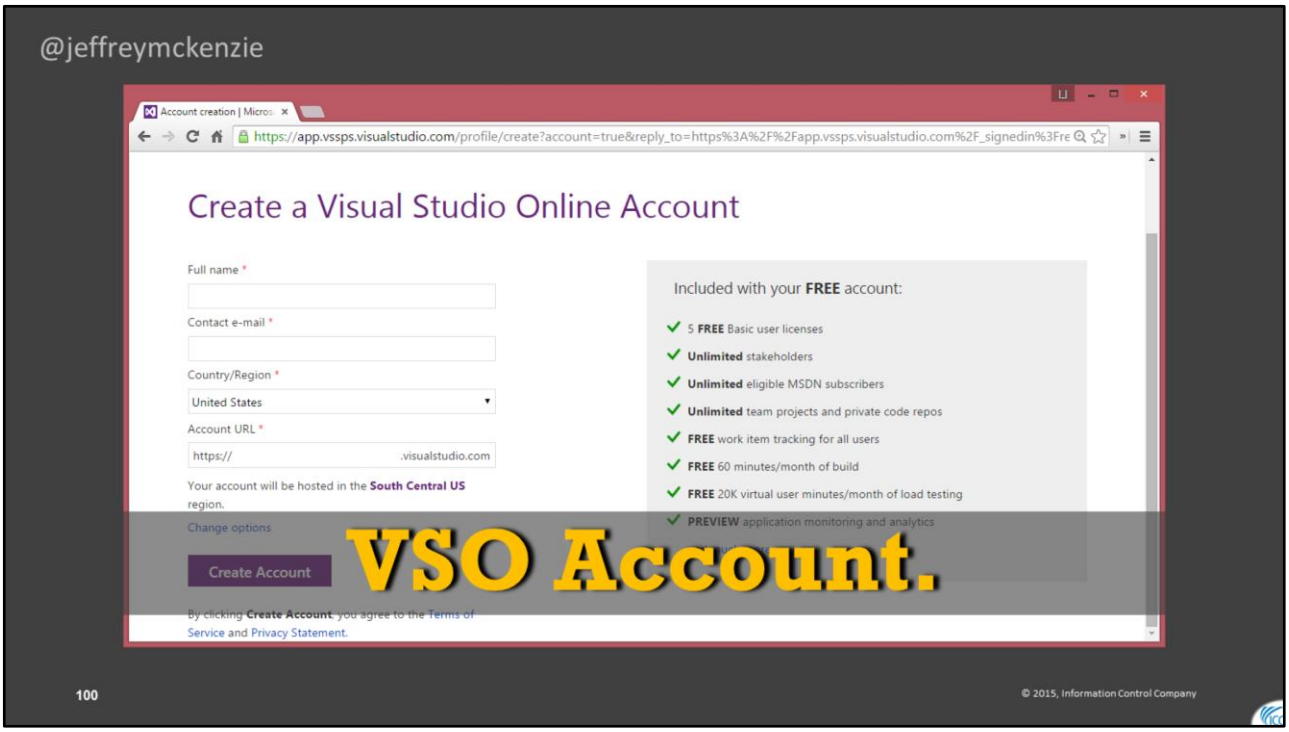
Instead of deploying TFS on a local server,
you can use the online service
without having to worry about
any of the infrastructure or hardware.



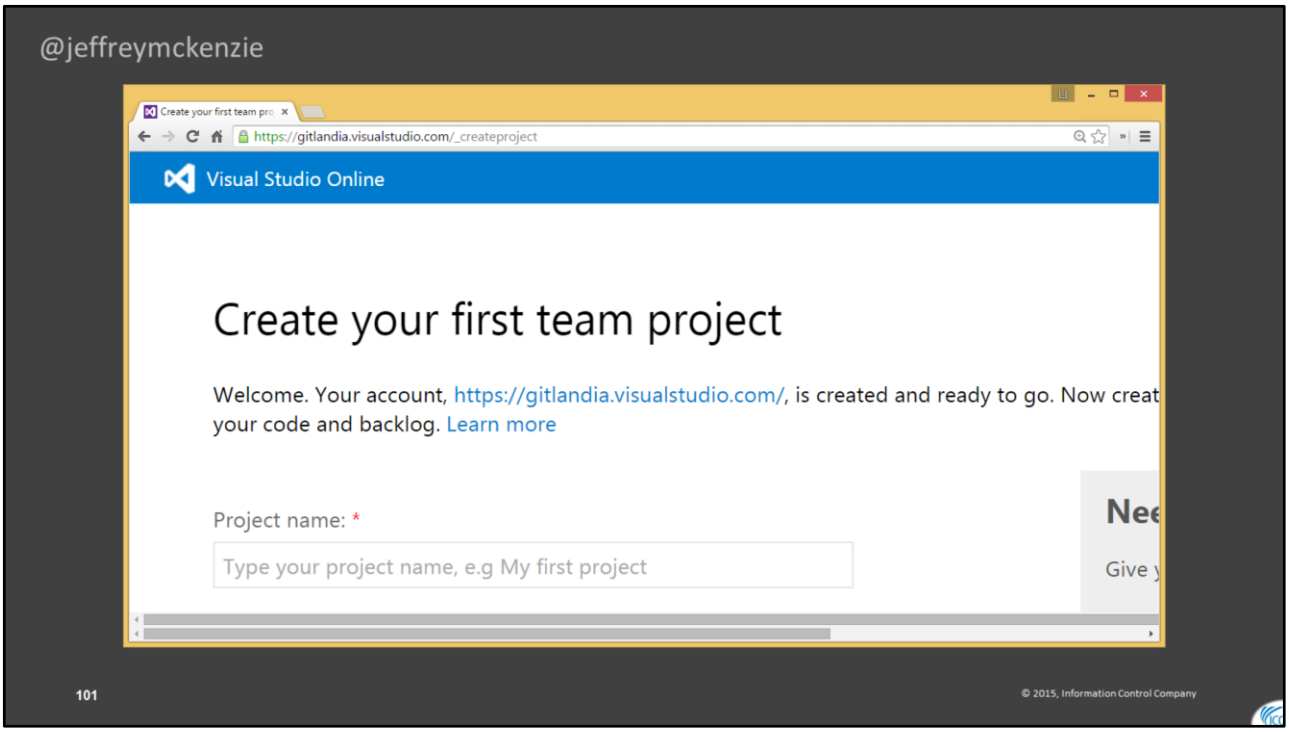
It's easy to get started, simply
Head out to visualstudio.com
And click on the signup link.
Almost all of the features I'm going
To be showing you are free for up
To 5 users, and it's also free if you
Have an MSDN subscription.



To use Visual Studio Online you do need a Microsoft account, so first thing is the prompt for MS account –
there's a link to create account
If you don't have one.



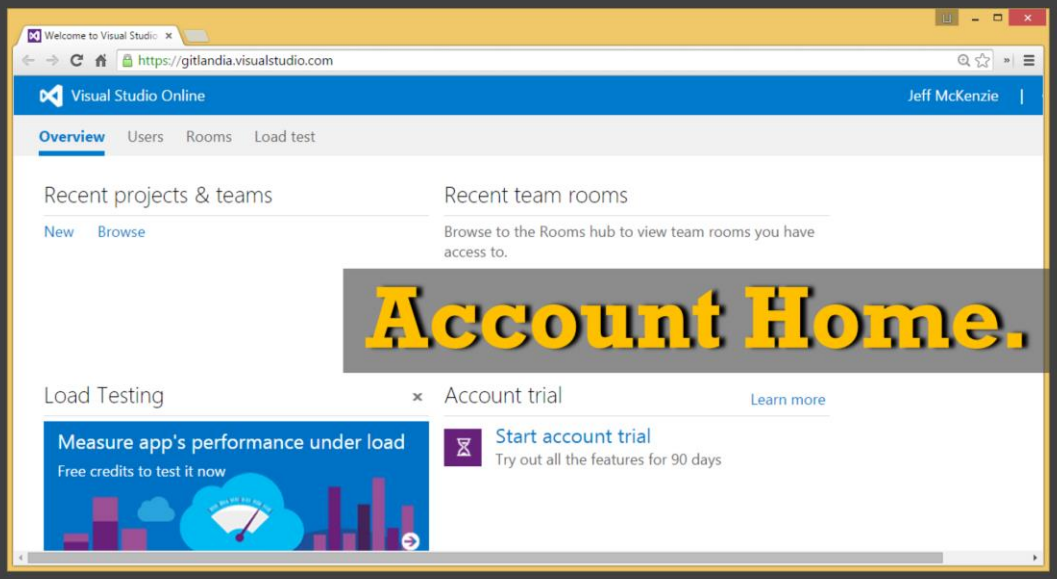
Once you create a Microsoft account, you can
Continue on to create a VSO account.
Very few details to fill in here, such as name,
email, and region. Since this is cloud-based,
the region determines which Microsoft data center
Hosts your data. Then it asks for your URL,
the domain name prefix you want to use.
So if you wanted, you could create….



Gitlandia dot visualstudio dot com.

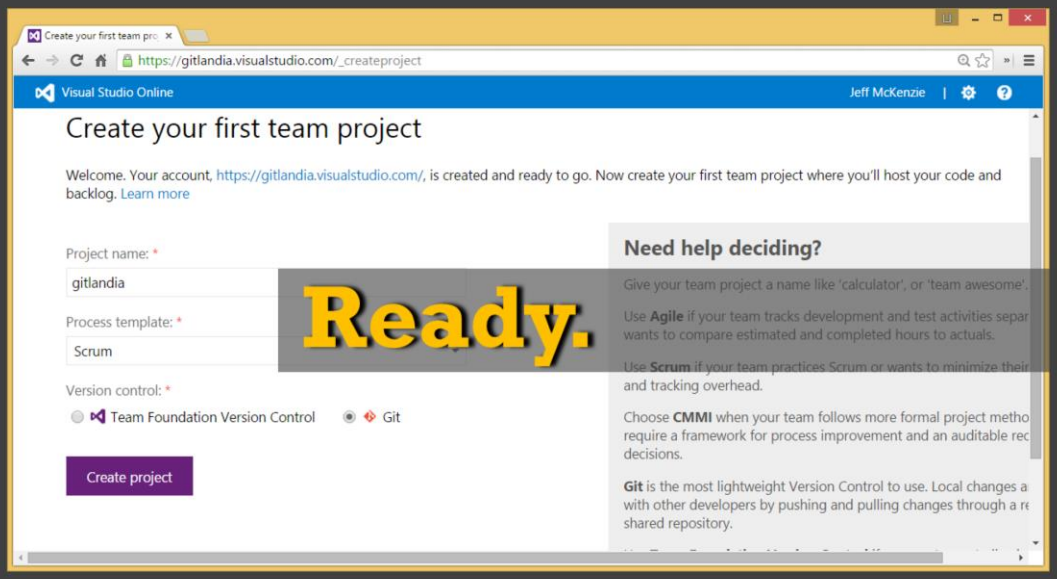
[image:]

@jeffreymckenzie



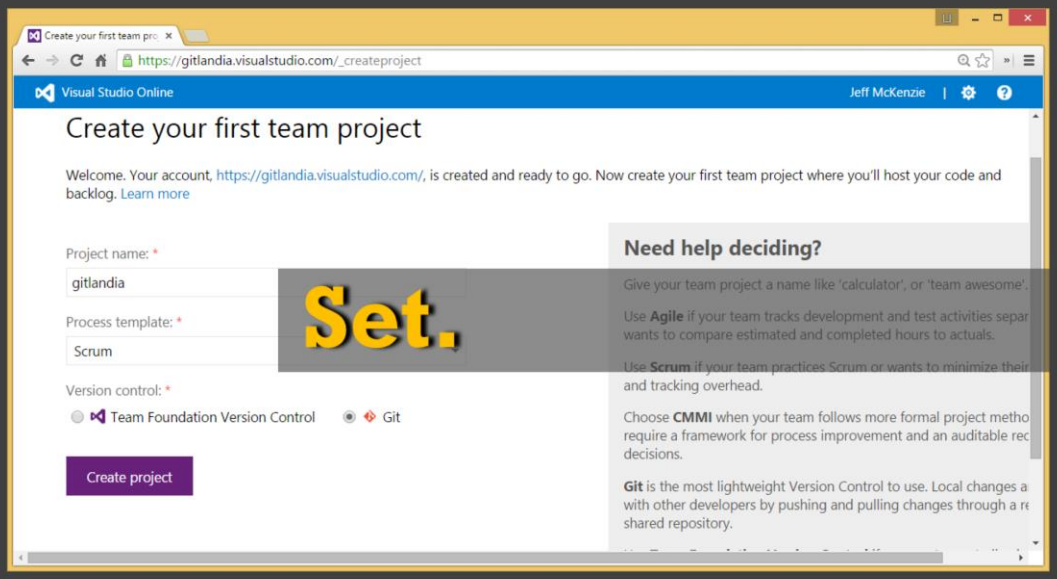
This is the home page for my own visual studio online account – under the recent projects and teams section below, I can create a new project.

@jeffreymckenzie



And that's it – a few steps and you're ready
to get started creating a project.
When you create a new team project
you have a few initial choices –
You select a project name and description –
here I've chosen gitlandia
And then you select your process template,
Which is essentially your project methodology of choice,
Such as scrum or agile.

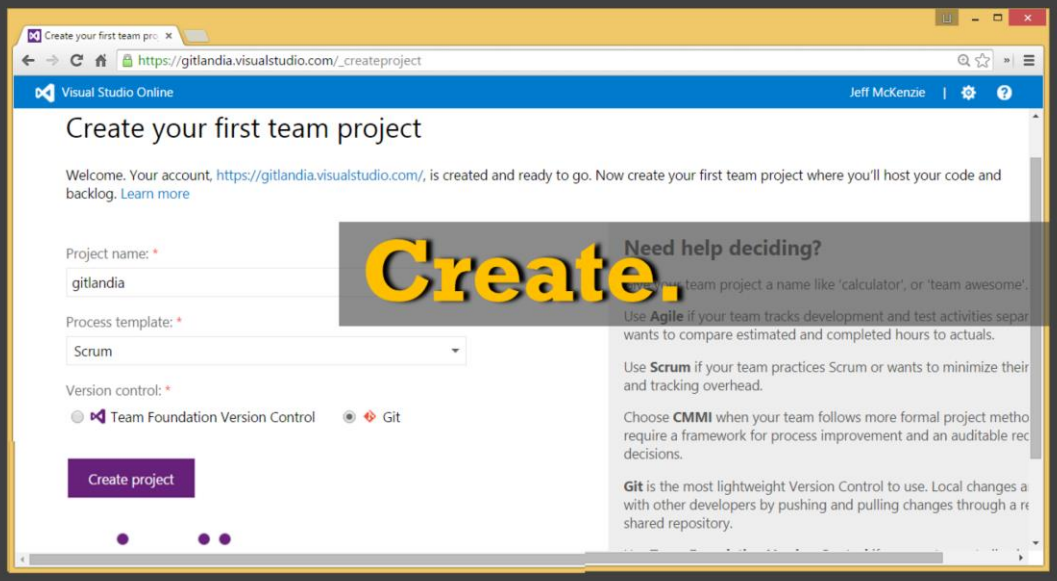
@jeffreymckenzie



104

Then you choose your version control –
either the traditional Team Foundation Version Control,
Which is centralized, or Git, Which is distributed.
In the past with TFS, you only had the option
of centralized version control.
In my opinion this is a huge change.
Many people were turned off to using TFS
because it has always used a centralized model.
But now, since you have the option
to go with distributed, this opens TFS and
Visual Studio Online to a wider audience.

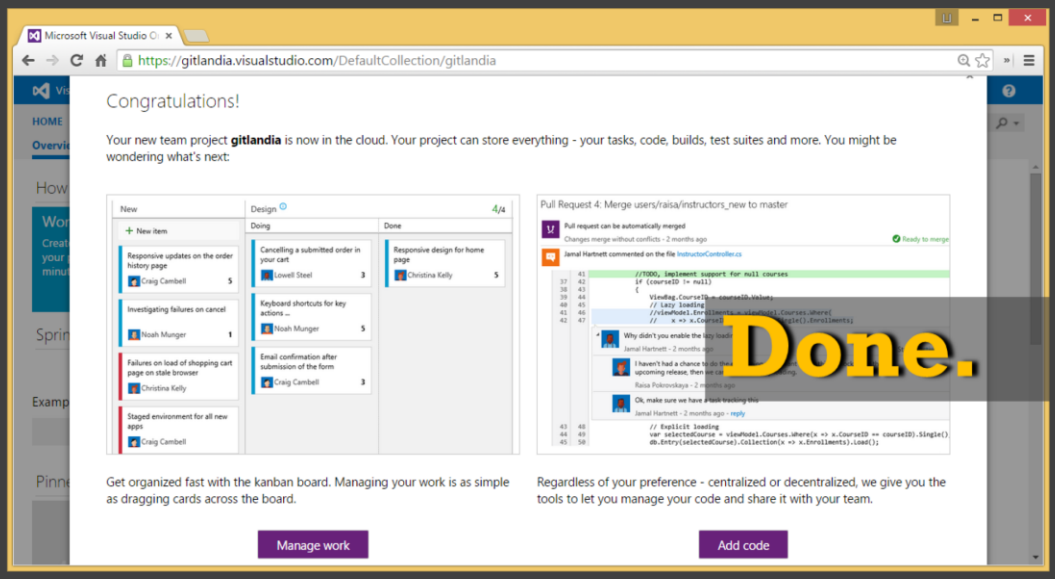
@jeffreymckenzie



105

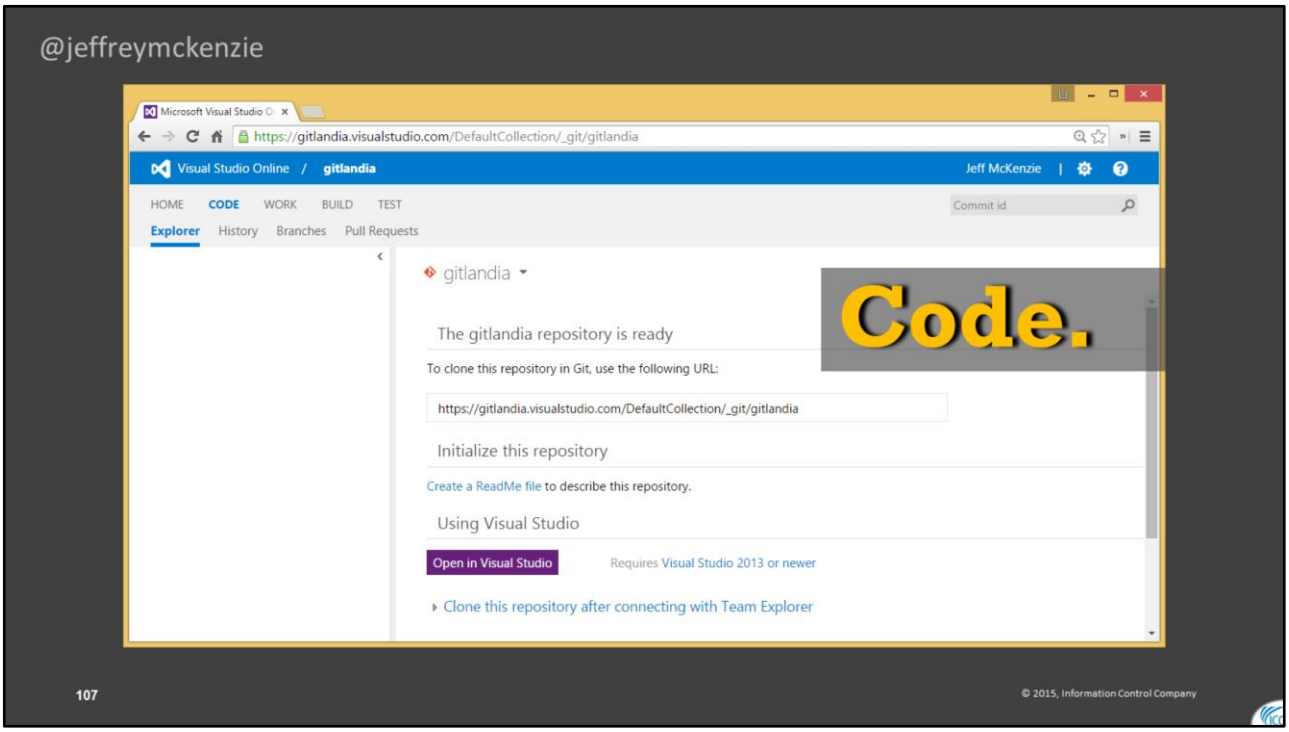
It takes a minute or so to get your
Project queued up and built
On the service….
And then

@jeffreymckenzie



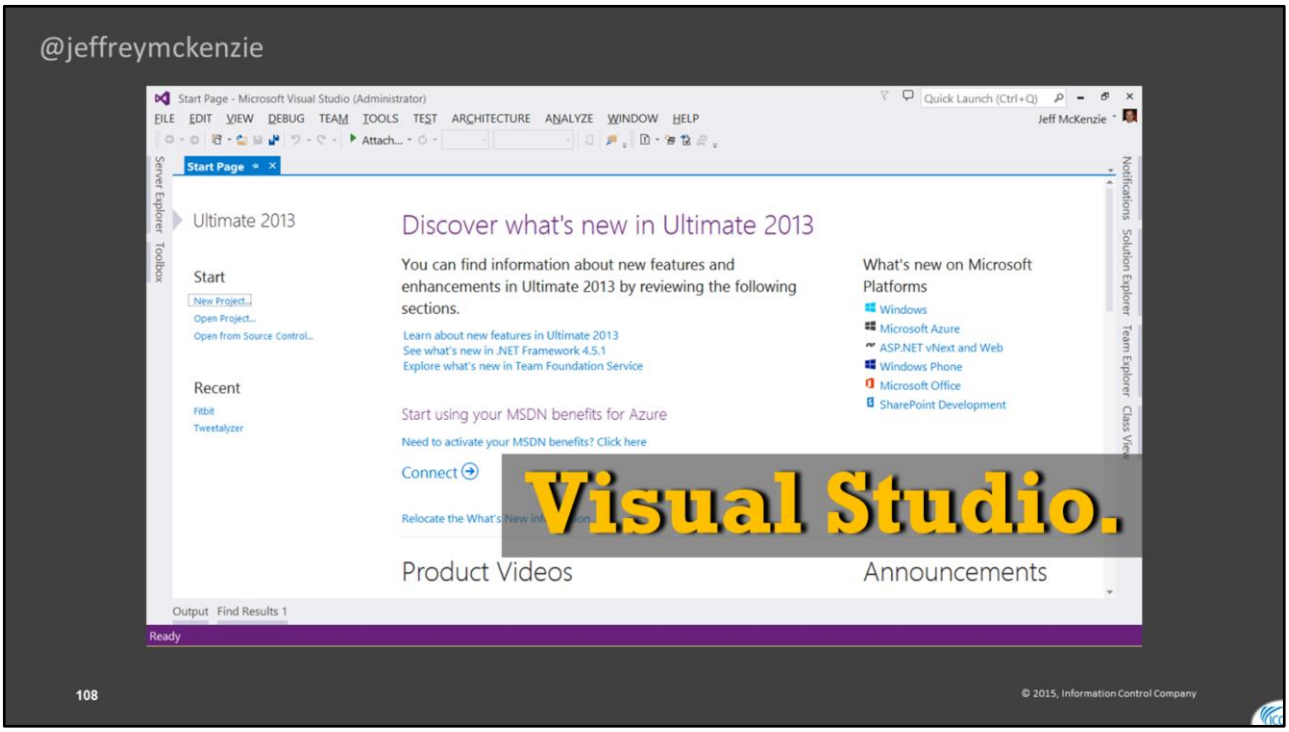
106

And when it's done it tells you that your project is now in the cloud
And gives you options on what to do next.



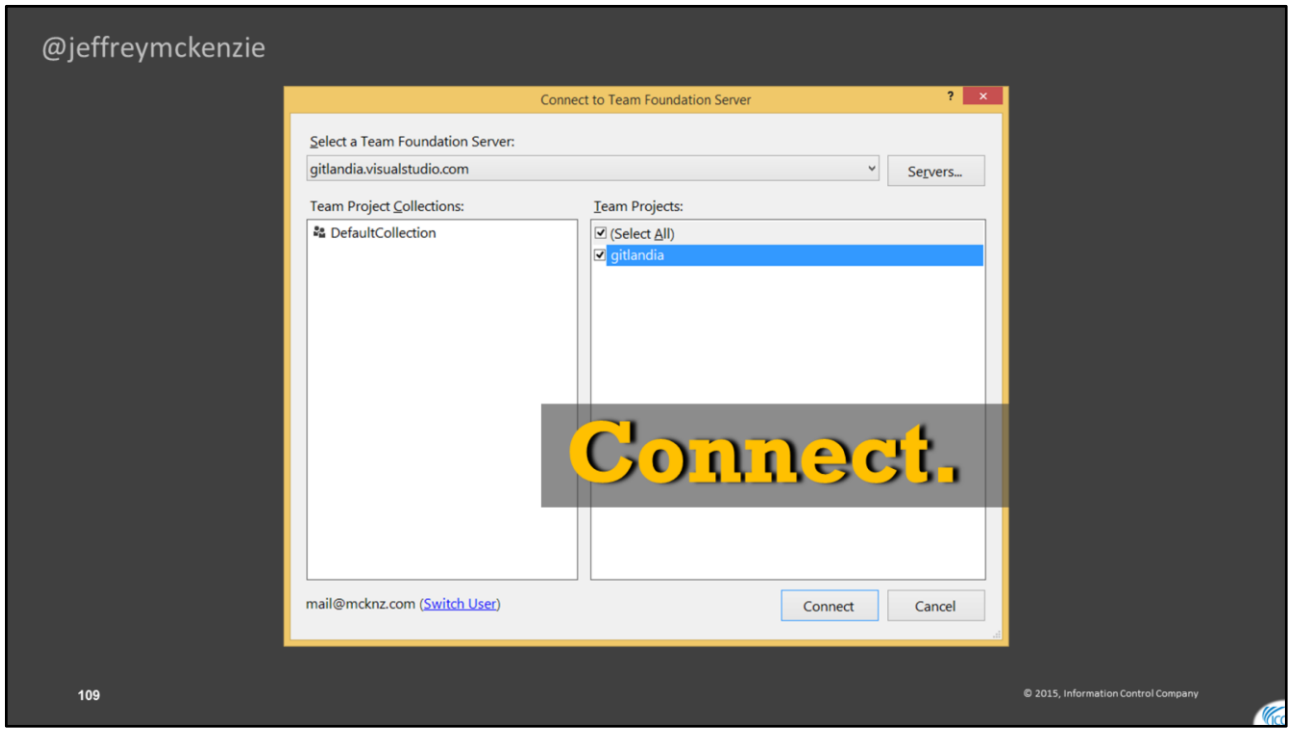
So I've called this project gitlandia, and toward
the top of the project page,
there are 5 main sections:
HOME, CODE, WORK, BUILD, TEST.
When we go to the Code section, we're shown
that the git repository for our project is ready, but empty,
which is expected here because we just created it.
We are given a couple of options on how to get started….

[image:]



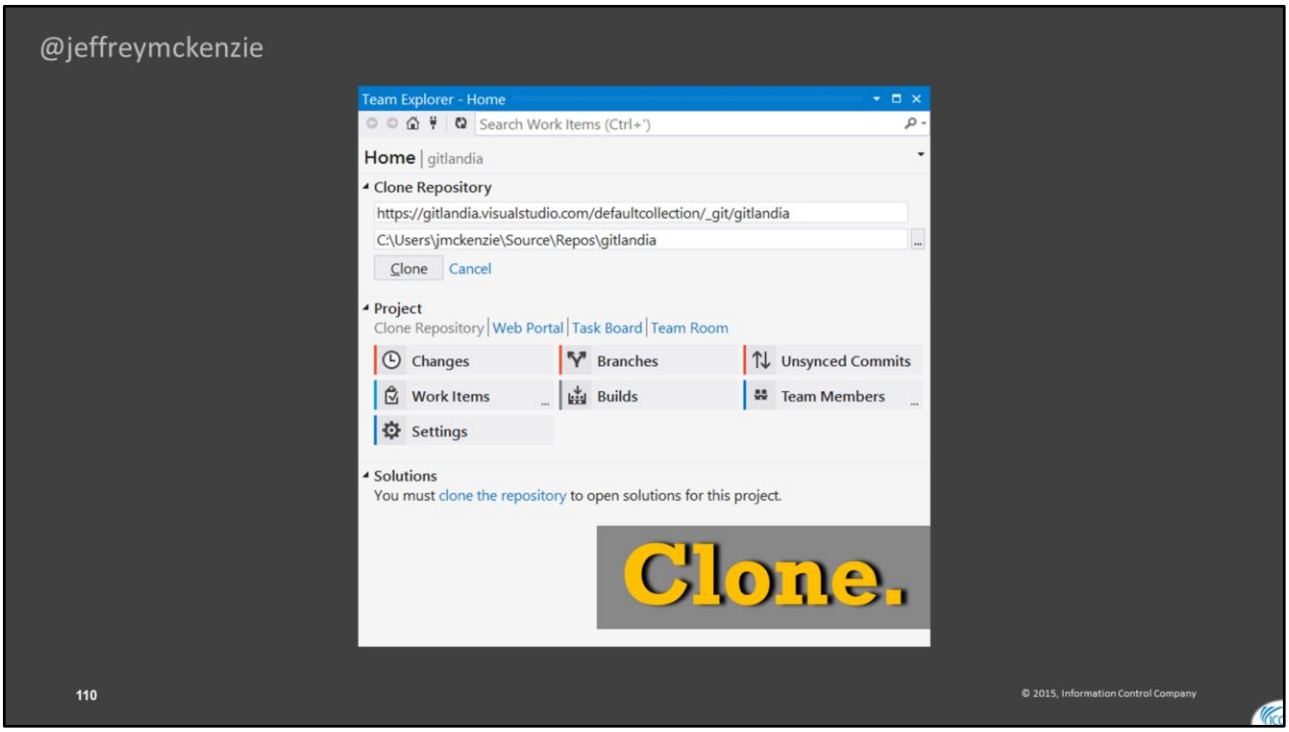
first, if you are using visual studio locally on your box, you can do what is called a clone of the repository, Which is essentially copying down all the files and revisions from the remote source. For git support, you'll need at least Visual Studio 2012 update 2.

[image:]



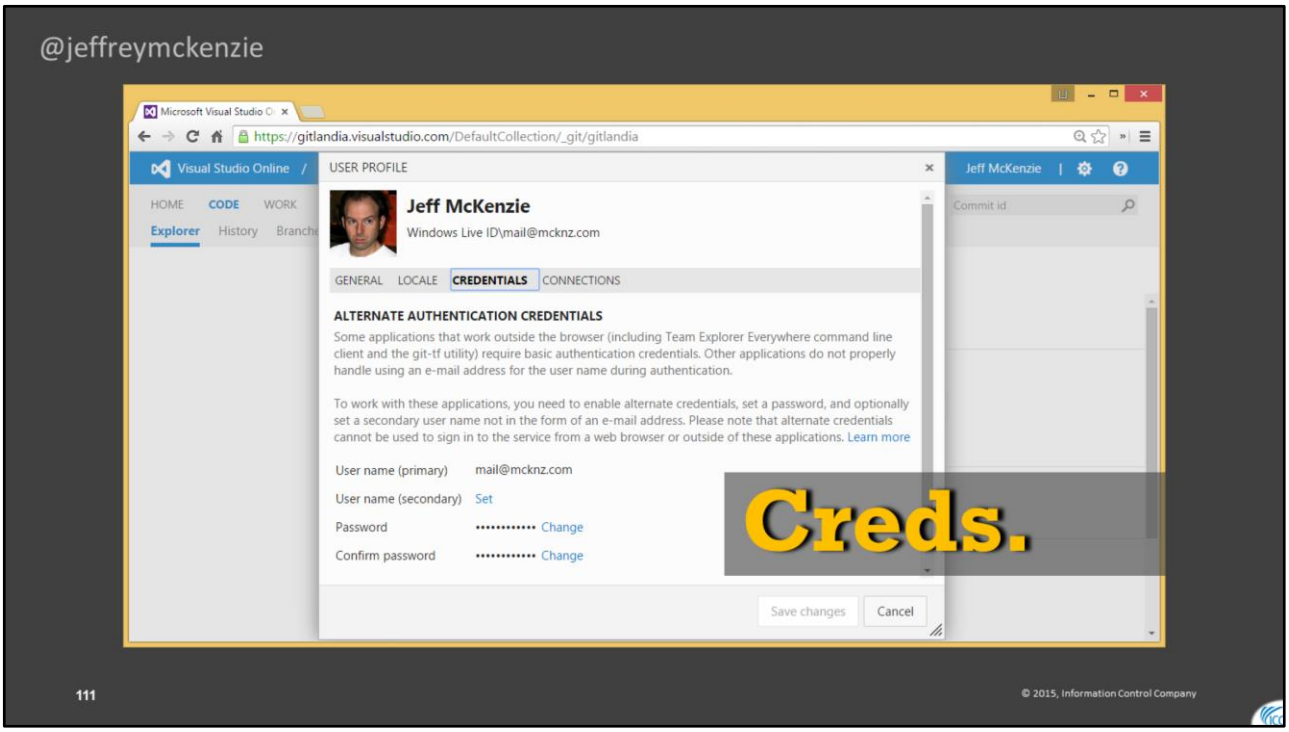
When you're in visual studio, you'll connect
to a team project just as you would with a local TFS,
Except here you'll enter the URL of your
Visual Studio Online account.
As you can see here, it shows the Gitlandia
project as available to me.

[image:]



In Team Explorer I'm prompted to enter the source and destination for the repository –
The local working directory
defaults to my user directory.
So working in Visual Studio it's pretty easy –
you connect to Visual Studio Online,
clone the repository, and you're ready to go.
It's slightly more complicated
with command line, but not much.

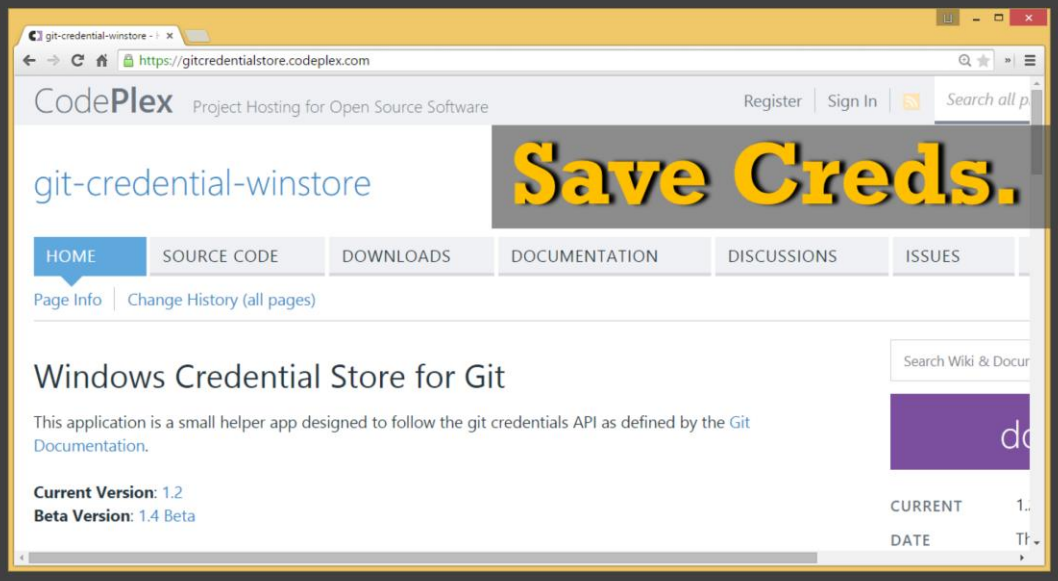
[image:]



If you're working directly connected to Visual Studio,
All the authentication is handled for you.

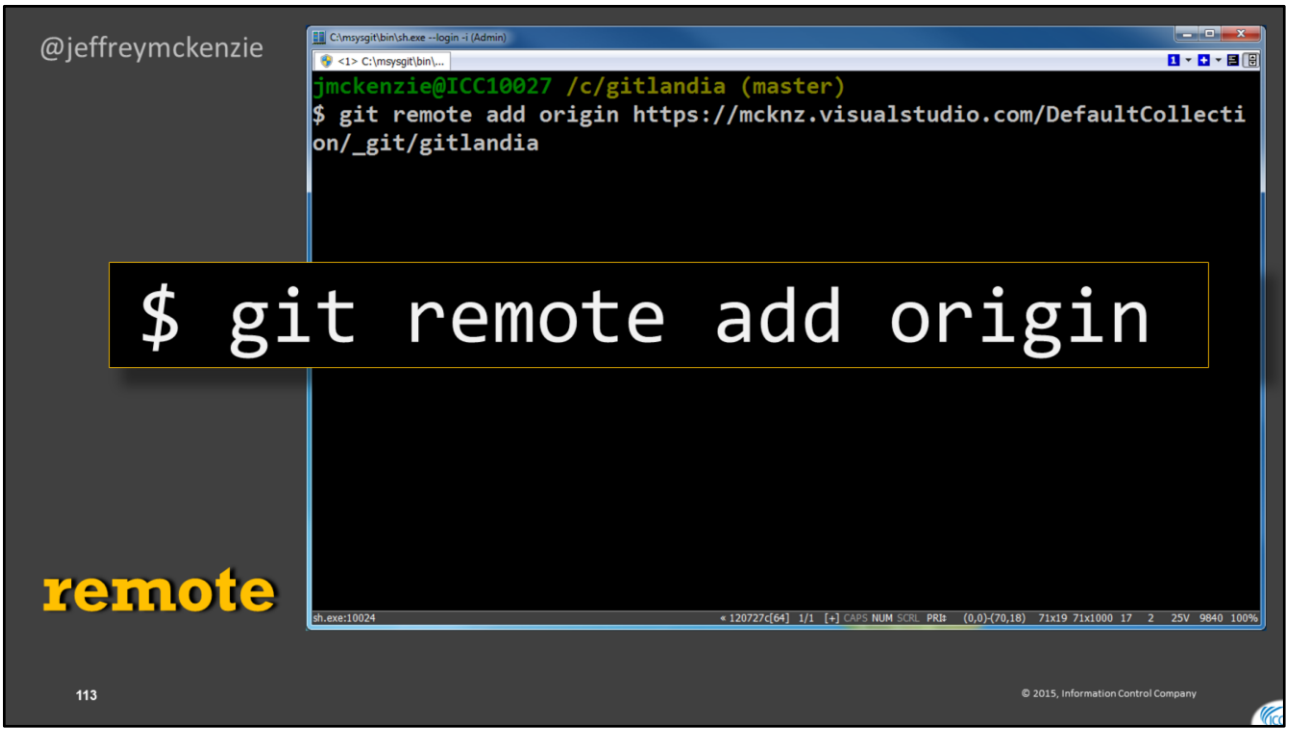
One thing you have to do in Visual Studio Online
to work on the command line is to enable
what they call alternate credentials.
You can just go to your profile online and set a password
so you can authenticate through the command line client.

@jeffreymckenzie



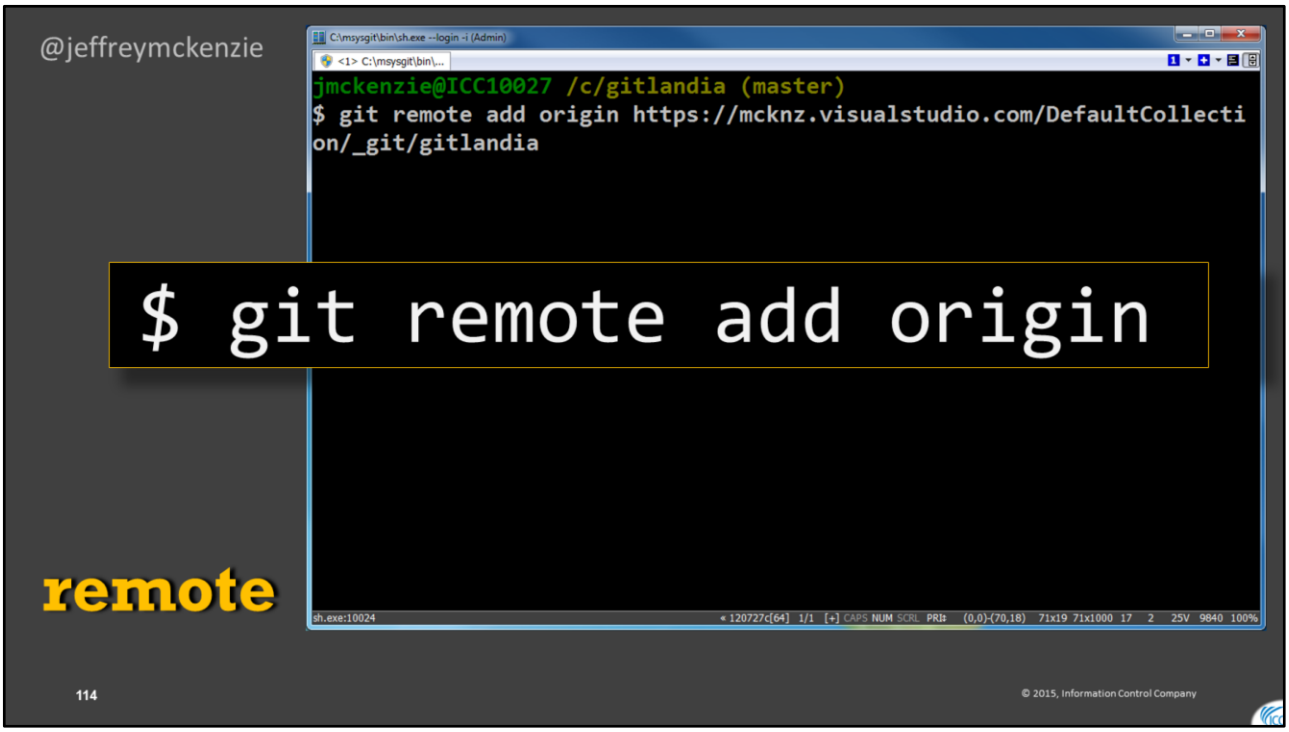
112

One of the problems about using the alternate credentials
Is that you have to keep entering them on the command line
Every time you push to Visual Studio Online.
If you are using Windows, you'll want to check out
This utility called git-credentials-winstore on codeplex.
This allows you to save your VSO username and password
In the standard Windows credential storage.



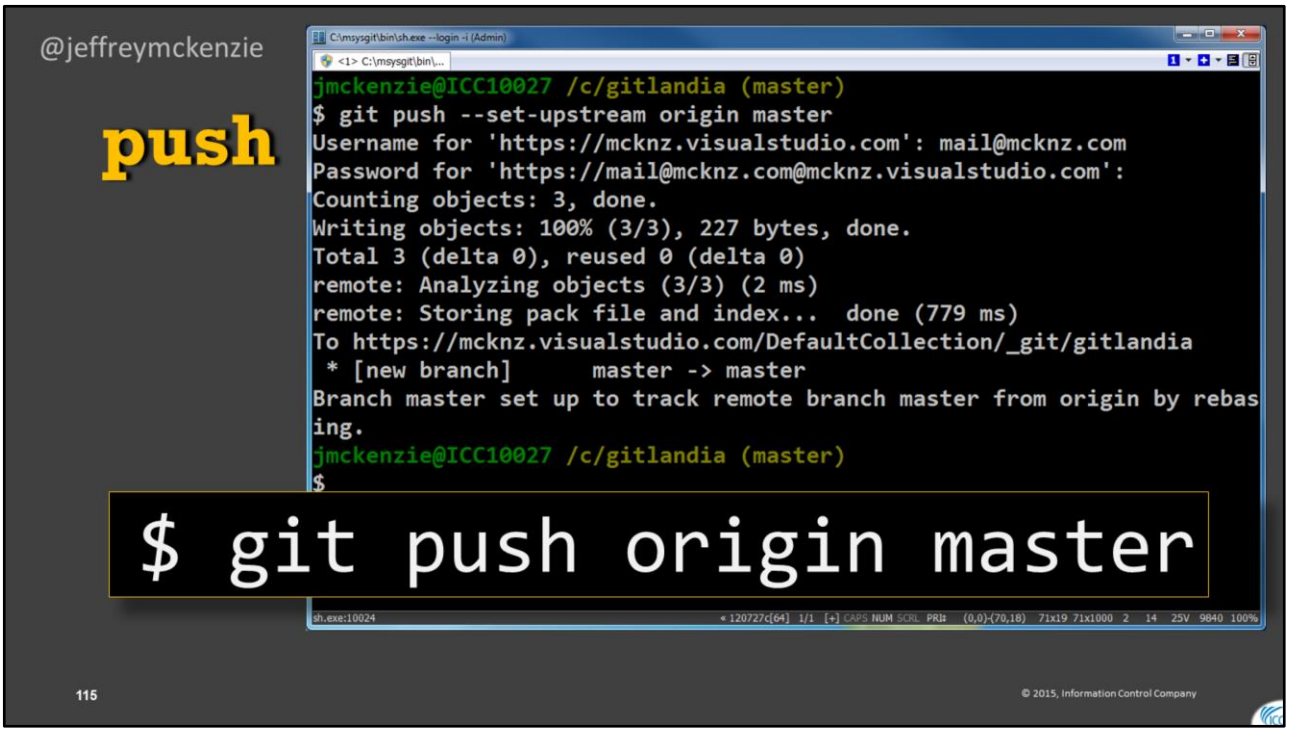
in git, there's the concept of a remote –
a remote repository is a reference to an
external copy of the source code.
In our case, the remote is visual studio online.

In order to create a link or reference
between my local copy
and visual studio online, I run the command
called "git remote add origin" –

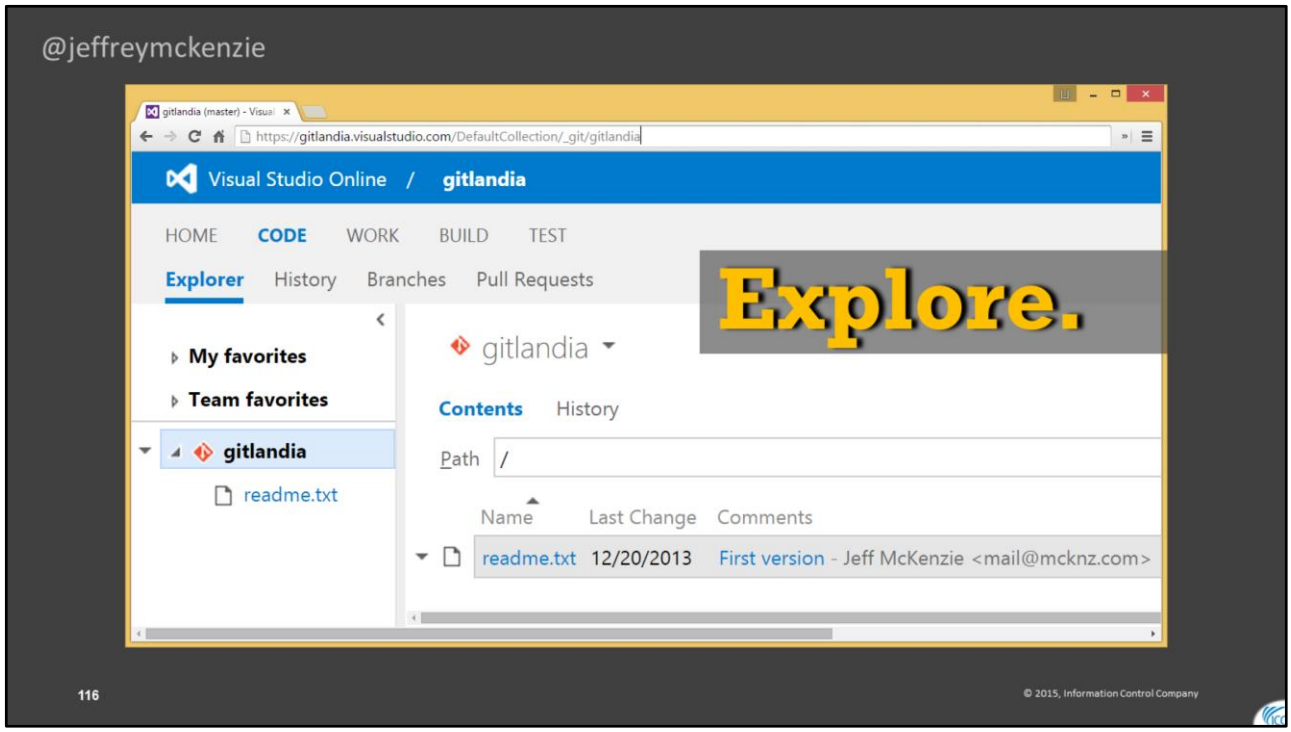


Which means I'm adding a remote called "origin" –
And origin is the name used by convention
to mean the external authoritative copy if the repo.

With that command, I also need to specify the URL
of the visual studio online account,
Which you can see here.
You can get that URL from the Code page
in Visual Studio online.

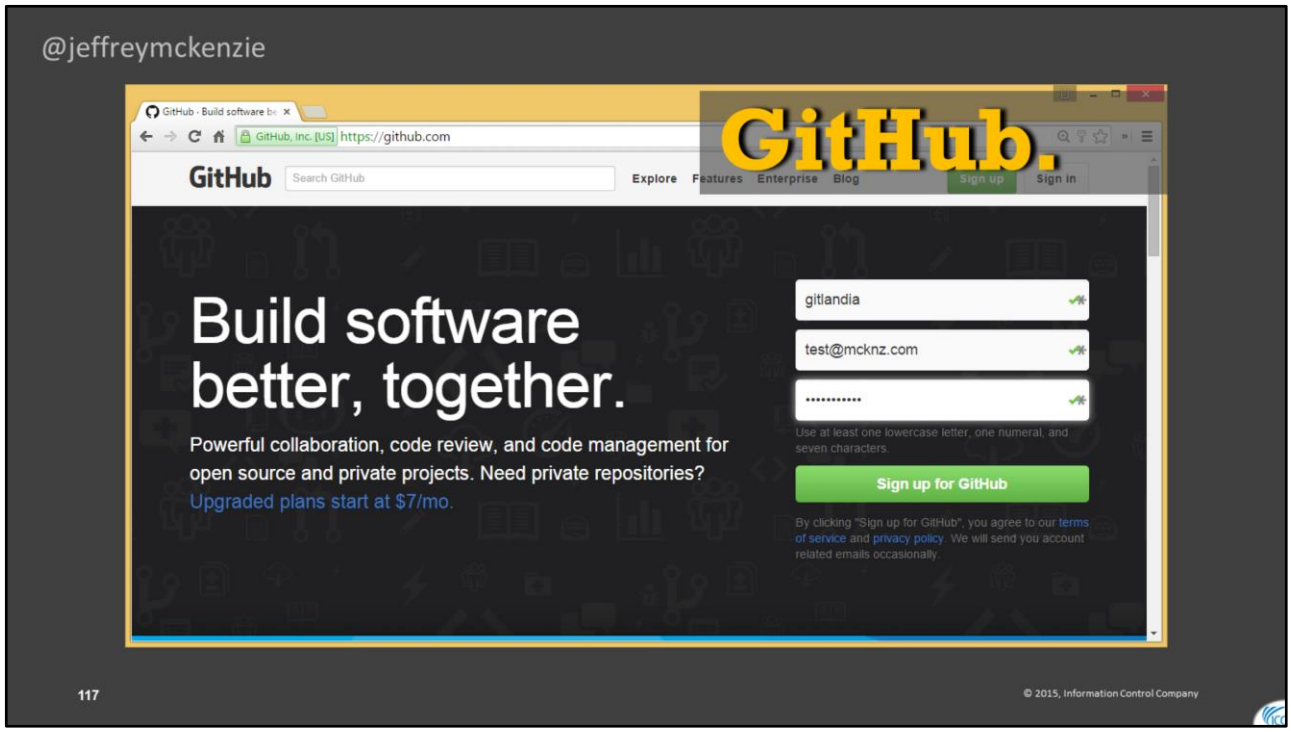


The push command allows you to move changes
to another git repository.
The full command is git push origin master.
And what that means is you are pushing to
the remote named origin,
which is visual studio online.
Master is the name of the branch
on the remote repository.



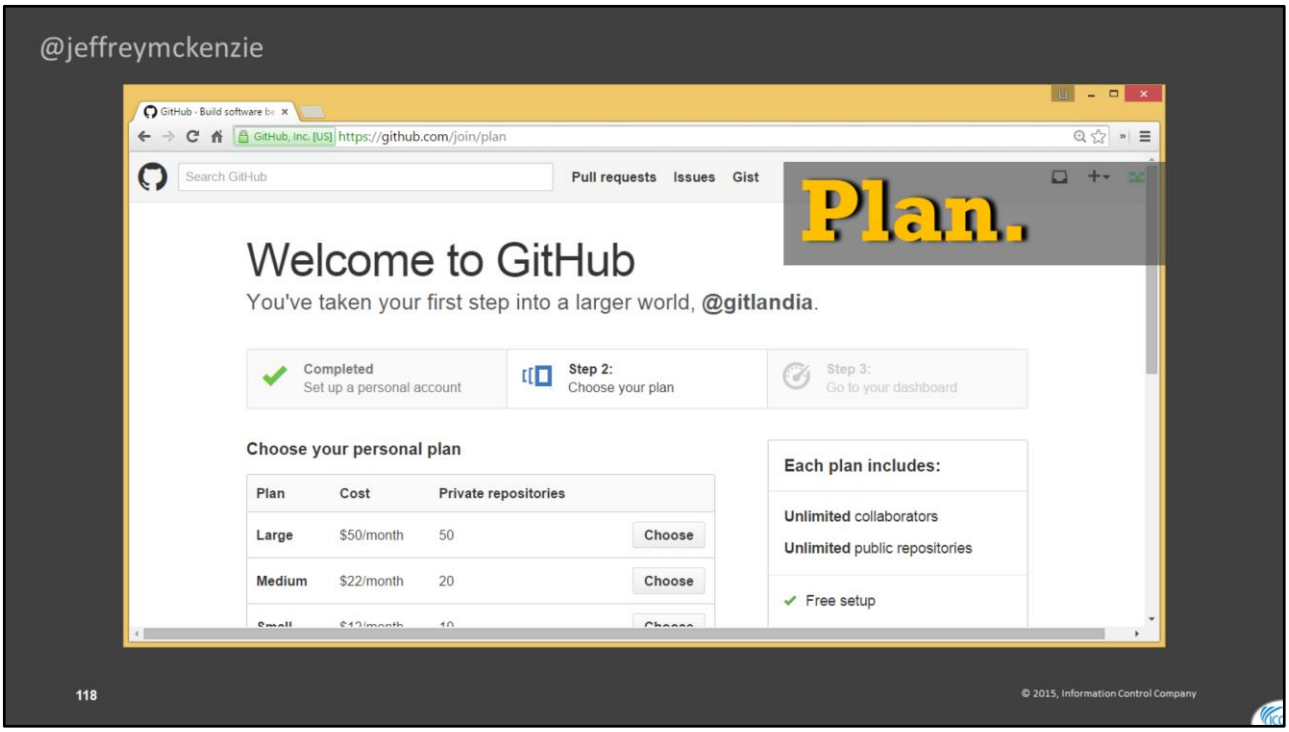
After the push, when you go back to Visual Studio Online,
you can see the updates through a graphical code explorer.

It no longer shows empty, but contains the readme file we created locally.



Next up is GitHub.
Signing up is quick and easy,
just enter an email, your user name
And a password to get started…

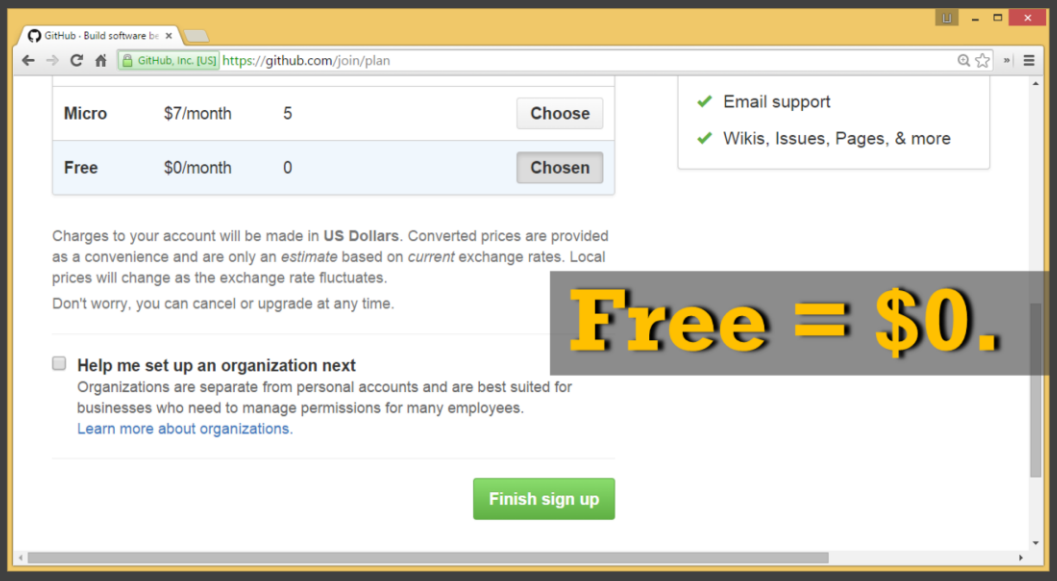
[image:]



Next you pick a plan,
Which you can choose
From the options available
GitHub is open and public
By default, so if you choose a paid plan,
You are paying for how many
Private repositories you want to have.

[image:]

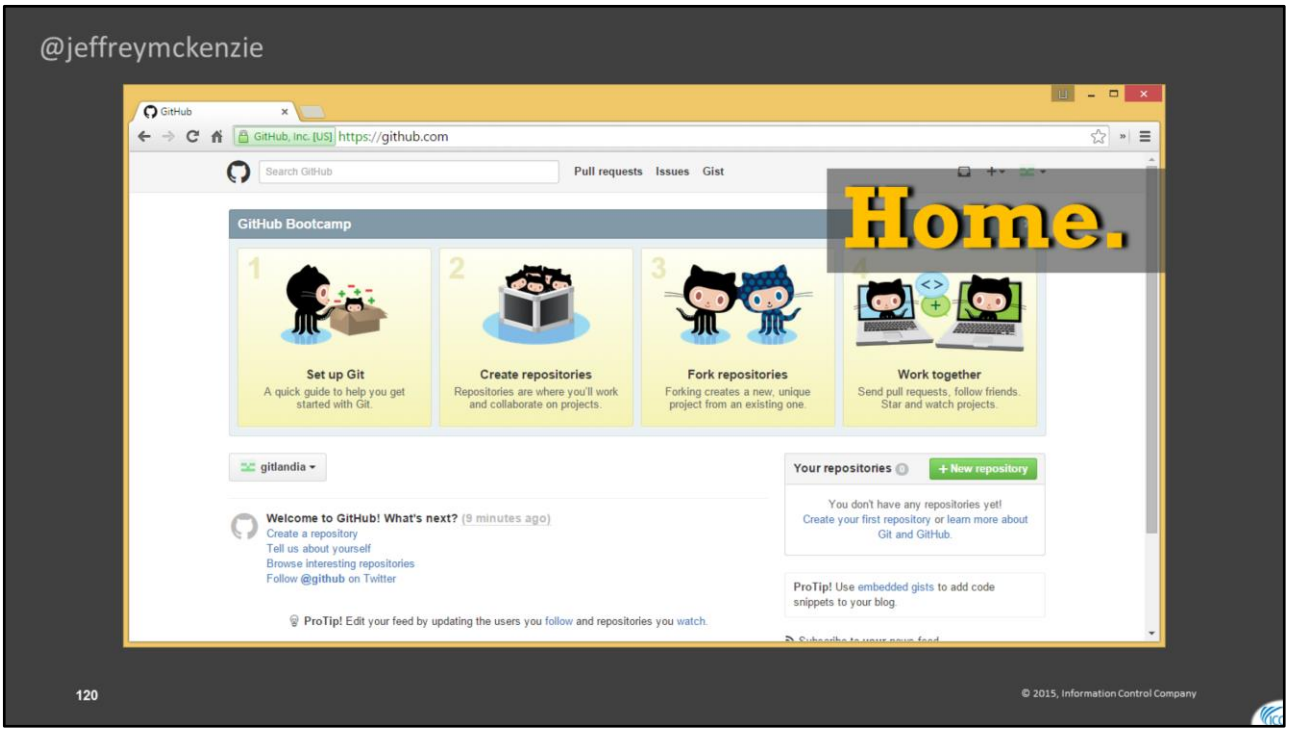
@jeffreymckenzie



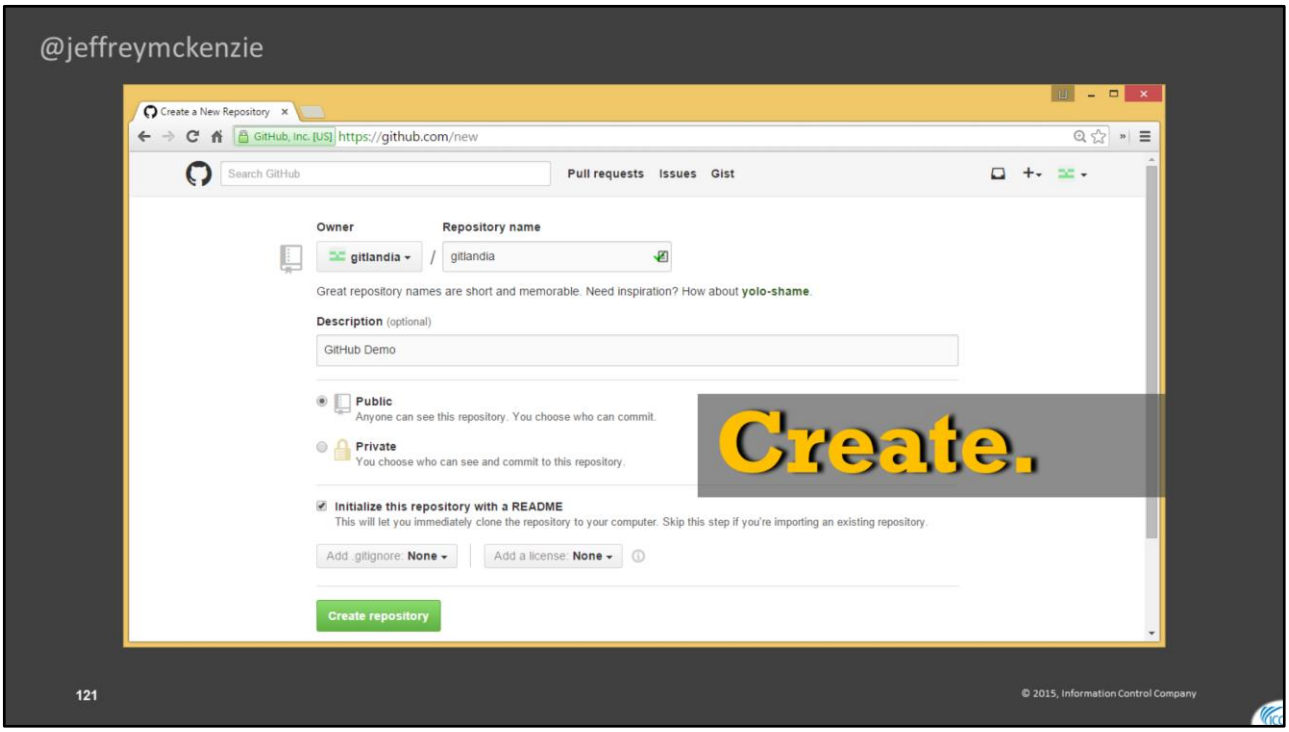
119

The free plan is, not surprisingly,
Zero dollars a month,
Which is auto-chosen for you.
You can set up an organization if you want,
Or you can go ahead and complete sign up.

[image:]

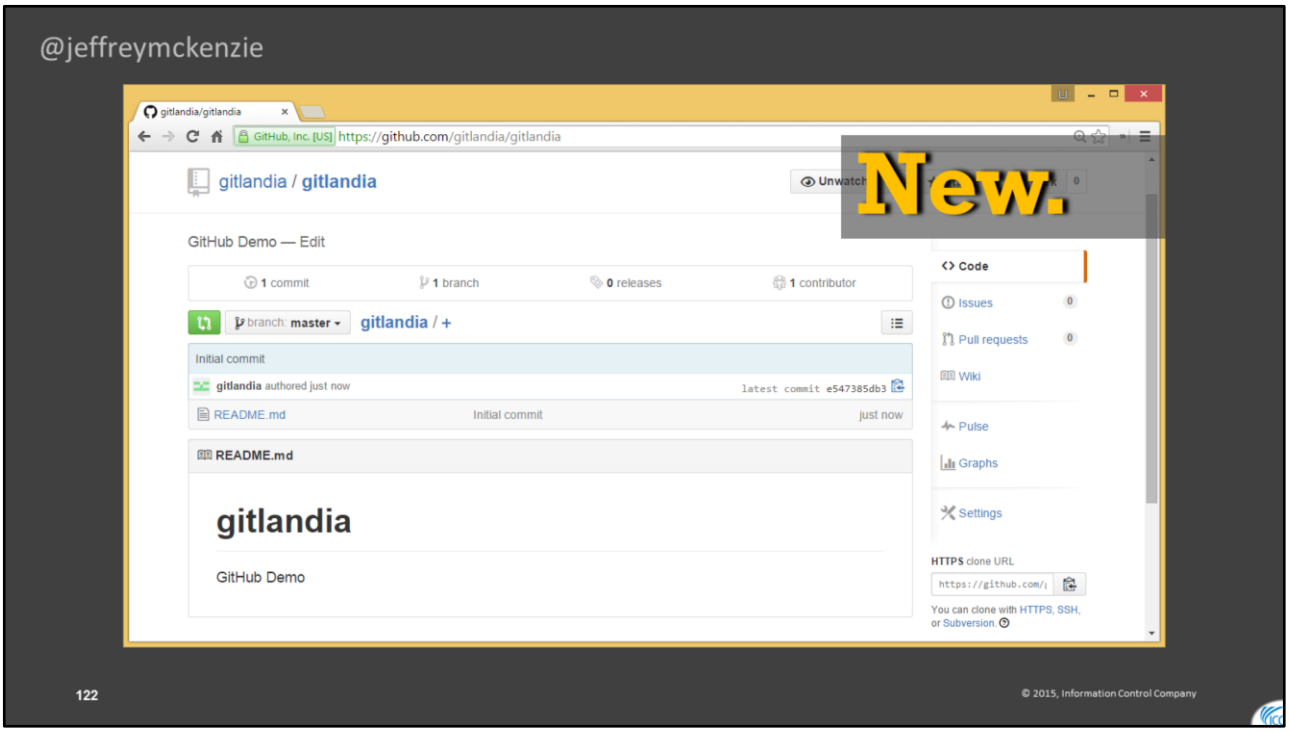


That takes you to the home page,
Where you are ready to start creating repositories.
A repository, or a repo, is simply a
Collection of files and their changes.
When I'm working in GitHub, I like to do
the opposite of the process we
just did in Visual Studio Online–
I create a new repo in github
and then pull it down to my local.

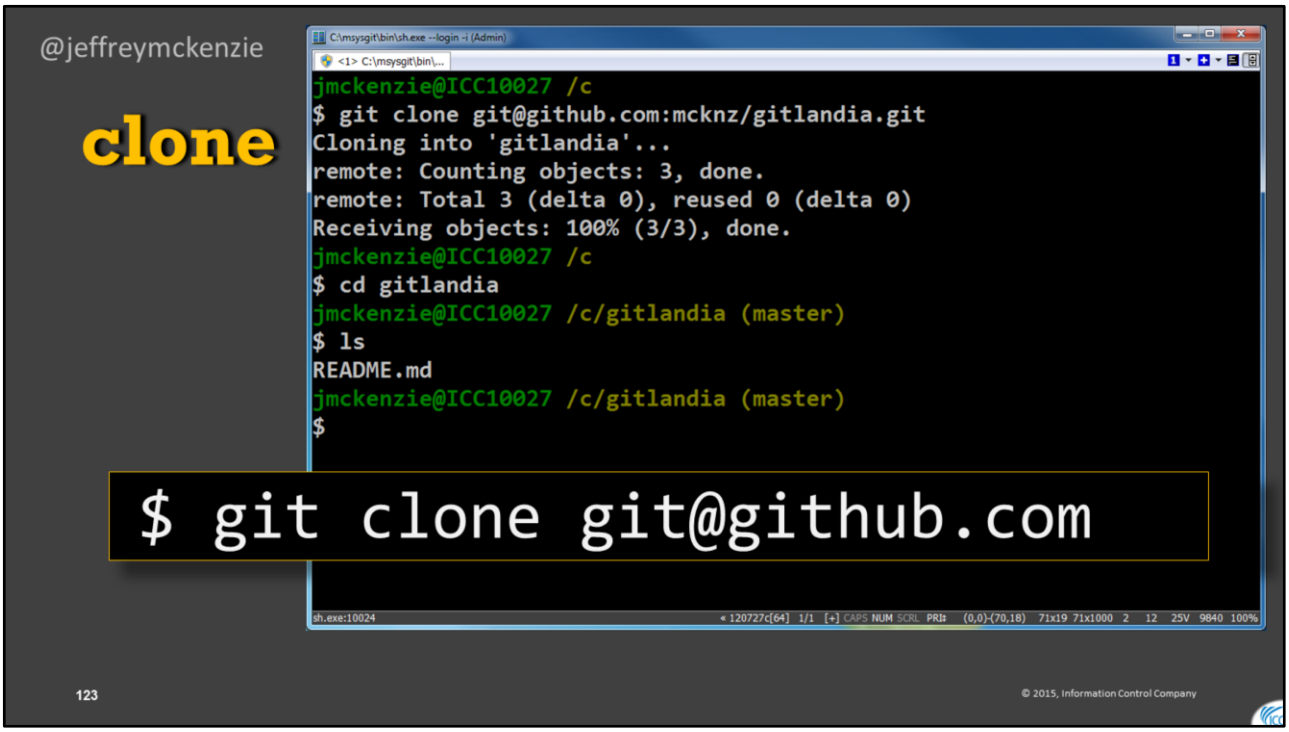


So let's create a repo called Gitlandia –
Github also gives you the option of initializing it
with a readme file –
That way you can clone it immediately.
If you try to clone an empty repository
it doesn't really do anything.

[image:]



So there it is, the new repository.
At the bottom right is a link I can use to clone it locally.
I use SSH with Github, which is a protocol
for secure communication that relies on a
public/private key pair.
It's nice because it creates a trusted relationship
Without usernames and passwords.



Back to the command line –
I've cleared out the old gitlandia directory
So I'm starting from scratch. I change directory
into the root C drive, and use the git clone command
With that SSH URL –
by default, when git does a clone,
it creates a directory that's the same name
as the repository

That's how easy it is to get started,
And that's the history of Gitlandia –

Thanks to you all –

What questions do you have about version control,
About the different models, or about Git,
GitHub, Visual Studio Online….?