Life of a Chromium Developer

tl;dr

Summary in command line form

Subversion Workflow

Get into work...

```
$ gclient sync
```

Hack on some files then create, upload and try a changelist...

```
$ gcl change my_new_feature
$ gcl upload my_new_feature
$ gcl try my_new_feature
```

Try server reports your patch failed because you forgot a file...

```
$ svn add forgotten_file.cc
$ gcl change my_new_feature
```

Iterate with reviewers, making sure to try your patches...

```
$ gcl upload my_new_feature
$ gcl try my_new_feature
```

Git Workflow

Get into work...

```
$ git pull --rebase && gclient sync
```

Hack on some files then create, upload and try a changelist...

```
$ git checkout -b my_new_feature
$ git cl upload
$ git try
```

Try server reports your patch failed because you forgot a file...

```
$ git add forgotten_file.cc
$ git commit
```

Iterate with reviewers, making sure to try your patches...

```
$ git cl upload
$ git try
```

Commit Workflow

LGTM! But hold before committing let's check the tree...

Red? Wait until tree goes green.

Green? Before committing make sure...

- ...your most recent try server attempt has passed
- ...you're in IRC or generally available on IM
- ...you're not leaving to catch a bus and will be at your desk

Good? OK let's commit!

```
(svn)$ gcl commit my_new_feature     (git)$ git cl dcommit
```

Don't commit and leave!

Handy Links

Documentation
dev.chromium.org

Bugs
bugs.chromium.org or crbug.com

Source
src.chromium.org

Buildbots
build.chromium.org

Code Reviews
codereview.chromium.org

IRC
#chromium on freenode.net

Mailing list
chromium-dev@chromium.org

Overview

Lots of information ahead!

Overview

How does one get involved in Chromium development?
Usually, you do some variant on the following work flow:

1. Get a machine that can build Chromium
2. Get the code
3. Modify and build the code
4. Test code
5. Upload and review code
6. Commit patch and waiting game

1. Development Machine

Because it takes a special kind of machine to build Chromium

Development Machine

Chromium is a large project!

- ~5000 build objects
- ~100 library objects
- 1 massive linked executable (~1.3GB on Linux Debug)

Why a massive executable?

- Easy to update!
- Faster to load!
- Hard to link :(

Even if you're building a 32-bit executable, you need a 64-bit machine since linking requires >4GB virtual memory.

Development Machine

General requirements:

- Lots of cores
- Lots of RAM
- Second hard drive for source code and building

Google Employees:

- Request machines and software from http://goto/stuff
- Apple software can be found at http://goto/apple

Windows:

- Z600
- Win7 64-bit
- Visual Studio 2008
- Follow Windows instructions

Development Machine

Mac OS X:

- Mac Pro
- Snow Leopard
- Xcode 3.2 or newer
- Follow Mac OS X instructions

Linux:

- Z600
- gold
- 64-bit Ubuntu Lucid or newer
- Follow Linux instructions

2. Getting the Code

Time to grab a coffee

The Source Tree

The source tree

- is the code required to build Chromium.
- consists of all the files under the src directory
- has files from many projects that Chromium depends on

Because the source tree has files from many projects, one full checkout of the source tree will actually contain a source tree for each of its dependencies.

The core Chromium code is only one part of this.

A Chromium checkout is actually a forest of source trees over multiple version control systems.

Getting the code - Basic steps

There are 3 steps:

1. Install Depot Tools
2. Create a .gclient specifying the main chromium trunk
3. Run gclient sync to get all the code and generate the build files

Detailed instructions can be found at:

http://dev.chromium.org/developers/how-tos/get-the-code

Refer to UsingGit and UsingWebKitGit if you're interested in using git over svn for Chromium and WebKit.

Installing Depot Tools

Installing depot tools consists of checking it out of svn, and putting the directory in your path. Detailed instructions are here.

Here's a condensed example (assumes linux):

```
  $ svn co http://src.chromium.org/svn/trunk/tooks/depot_tools
  $ export PATH=`pwd`/depot_tools:"$PATH"
```

The first command creates a subdirectory "depot_tools" with a checkout of Depot Tools.

The second command places the utilities in your path. Note because it uses `pwd`, it should be invoked in the directory above "depot_tools."

What Depot Tools Does

Depot Tools is a set of scripts/utilities that:

- Manage all checkouts in the Chromium source tree
- Generate the build files for your platform
- Upload your changes to Rietveld for review

Common utilities:

- gclient - syncs your source tree and creates build files.
- gcl - bridges SVN to Rietveld to publish changes for review
- git-cl and git-try - Gives similar functionality to gcl, but for git

Depot Tools Quirks

Common Quirks:

- The Depot Tools installation is actually an svn checkout, even if you got it from a tar or zip file
- gclient attempts to do an `svn up` each run and can fail here
- gclient sync both updates your source code, and regenerates build files (implicitly doing `gclient runhooks`)

Creating a .gclient

The .gclient file specifies where to get the core Chromium code.

This command (all on one line) will create a default .gclient:

```
  gclient config
      http://src.chromium.org/svn/trunk/src
      http://chromium-status.appspot.com/lkgr
```

Read get the code setup for most up-to-date instructions.

What this .gclient does

The default .gclient will look like:

```
solutions = [
  { "name" : "src",
    "url"  : "http://src.chromium.org/svn/trunk/src",
    "safesync_url" :
        "http://chromium-status.appspot.com/lkgr",
  },
]
```

What this does:

- Gets the "lkgr" from safesync_url
- Creates a subdirectory named src if it doesn't exist
- Does an svn checkout of the url, at the "lkgr".

lkgr = "Last Known Good Revision", the last revision to pass all the automated build tests.

Running gclient sync

This step is easy. From anywhere within the source tree, run:

```
gclient sync
```

This will download the initial repositories listed in .gclient (if you aren't using git), find all the DEPS files, and then download all the source code listed in those.

This can take a few hours. Grab a coffee.

For best results check out the code on a dedicated hard drive and avoid NFS drives at all costs.

gclient sync Gotchas

Only 2 Gotchas:

1. gclient sync **does not update your git checkout**
2. gclient sync does not download the same code on each platform

As an example of #2, this directory will not exist in a windows checkout:
src/third_party/WebKit/WebKit/mac

Looking in src/DEPS, this is listed in the mac-only section.

And you're ready to build!

If you've made it this far, you should have all the code needed to build chromium. Here's a cookbook of the commands

1. Install Depot Tools

```
svn co http://src.chromium.org/svn/trunk/tools/depot_tools
export PATH=`pwd`/depot_tools:"$PATH" # or windows equiv
```

2. Create a .gclient

```
gclient config

    http://src.chromium.org/svn/trunk/src
    http://chromium-status.appspot.com/lkgr
```

3. Sync the source code

```
gclient sync
```

3. Modifying and Building

Development Environment (IDE, etc.)

There is no single supported platform, IDE, or source control system. Use what you're comfortable with!

Common Setups:

- Linux: vim/emacs and git
- Mac: Xcode and git
- Windows: Visual Studio and SVN

For code completion: Google Code Search :)

Building Code: Unix

In the **src**/ directory:

```
$ make chrome
```

Specify build type (default is Debug)

```
$ make chrome BUILDTYPE=Release
```

Parallel build across ## machines or processors
(do this! will take hours otherwise)

```
$ make -j## chrome
```

Builds to **src/out**/ directory, e.g. **src/out/Debug/chrome**

For clean build, delete **out**/ directory (rarely needed)

Building Code: Xcode

In the **src/chrome/** directory:

```
$ xcodebuild -project chrome.xcodeproj -configuration Debug -target chrome
```

Can also build from Xcode.

Builds to **src/xcodebuild/** directory, e.
g. **src/xcodebuild/Debug/Chromium.app/Chromium**

Other targets are stored in **.xcodeproj** files in relevant folders

Building Code: Visual Studio

Launch Visual Studio cmd.exe environment and in the **src/chrome/** directory:

```
$ devenv.com /build Debug chrome.sln /project chrome.vcproj
```

Can also build from Visual Studio.

Builds to **src/chrome/** directory, e.g. **src/chrome/Debug/chrome.exe**

Other targets are stored in **.vcproj** files in relevant folders

Common Build Problems

"make: * No targets specified and no makefile found"**
You need to run gyp to generate Makefiles. This is best done via gclient (running `gclient sync` does this automatically):

```
$ gclient runhooks  # from anywhere under src/
```

Random C++ errors from a fresh copy of source
Check waterfall to see if the tree is red. If so, try updating the code when the tree is green and build again.

Confusing link errors on a checkout that used to work
Try deleting your output directory to force a clean build. If using Visual Studio, make sure you quit before syncing.

Debugging

Debugger needs to attach to browser **and renderer** process.
But by default, only the browser process is attached...

To attach renderer process, either:

- Run with --single-process flag
  (easier, but unsupported and known to be buggy)

- Manually attach renderer process to debugger
  (trickier; instructions depend on debugger)

See platform-dependent pages for more details: linux, mac, win

Making Good Changes

Find a bug via the Chromium bug tracker or grep the source code for TODOs to find tasks to work on.

NOTE: Just because a bug is filed doesn't mean it should be fixed. Try to check with senior project members before spending time making big code and/or UI changes.

The Chromium design docs provide high-level explanations of the architecture for various components of chrome

Use Code Search to quickly search through the entire project.

Follow the Chromium code style guide!

4. Testing

Testing Overview

Chromium has **a lot** of tests!

Roughly broken down into following categories:

- Unit tests
- UI tests
- Performance tests
- Layout tests

We also run some tests through Valgrind to catch memory and threading bugs.

In general, developers submit patches to the try server to build and fully test changes on all platforms.

Testing Overview

Most tests are written in C++ using gtest and are hosted in Chromium's repository.

Layout tests are written in HTML/CSS/Javascript and are hosted in WebKit's repository.

It's typically enough to build the tests for the code you're modifying and run them as opposed to the entire test suite.

Ask your team mates which tests you should build and run!

Try Server

The try server is the easiest way to test your change on all platforms. **Highly recommended!**

The try server takes your change, applies it to a clean checkout of the source, compiles **every possible target**, then runs **most** (not all!) tests for each platform, and emails you the results.

To submit your change to the try server:
Subversion: gcl try [changelist name]
Git: checkout branch you want to try and run git try

Requires commit access!
You may also email nsylvain@chromium.org
for try server access

Try Server

However the try server doesn't work for all types of changes:

- Applying SVN properties
- Binary content
- Patches containing CRLF characters

It still works for the vast majority of cases, **so use it!**

It's bad form to commit a change that breaks something because you didn't feel like waiting for try server results.

Read try server usage for more information.

Flakiness

Sometimes the tests fail for unrelated reasons. It usually happens for larger tests, which have a lot of reasons to fail:

- there might be some problem on the machine running tests
- the browser may crash randomly
- there may be a race condition in the browser or test

Flaky failures are bad. We shouldn't need to wonder whether a given test failure is real or flaky. A test failure should always mean broken code.

Fighting flakiness

- File bugs for intermittent test failures.
  Use Tests-Flaky bug label.

- Write solid test code.
  - Would you use a Sleep in browser code to "wait" for events? Then don't use it in tests.
  - Most test infrastructure helpers should be synchronous. For example, when NavigateToURL returns, the navigation is finished.
- Write smaller tests.
  - Start with a unit test. Only if needed, use a UI test.
  - In-process browser tests are good compromise: they launch a full browser, but you can access all the internals in the same process.

5. Uploading for review

Why code reviews?

Chromium has a large, complicated code base with many layers of abstractions that paper over tricky IPC, threading, and resource management semantics.

Reviews are done by uploading your change to Chromium's Rietveld instance at http://codereview.chromium.org

All code should be reviewed prior to checkin.

Uploading code

Once you're happy with your change and tested it, you're ready for review!

If you have never uploaded code before, go to Rietveld and log in with your chromium.org account or a Google account of your choosing.

Upload your change using the following:
Subversion: `gcl upload [changelist name]` Git: checkout appropriate branch and run `git cl upload`
If prompted, enter the same credentials you used to log into Rietveld above.

Uploading code

Write a **meaningful and descriptive changelist description** and fill in the BUG= and TEST= fields. If a bug doesn't exist, go ahead and file one just for your change.
View your uploaded change in Rietveld (URL should be printed to console after running `gcl upload`/`git cl upload`).

Finding reviewers

Best to ask your team mates or inspect commit logs:

```
svn log/annotate [path]
git log/annotate [path]
```

Use "annotate" links on files at http://src.chromium.org/

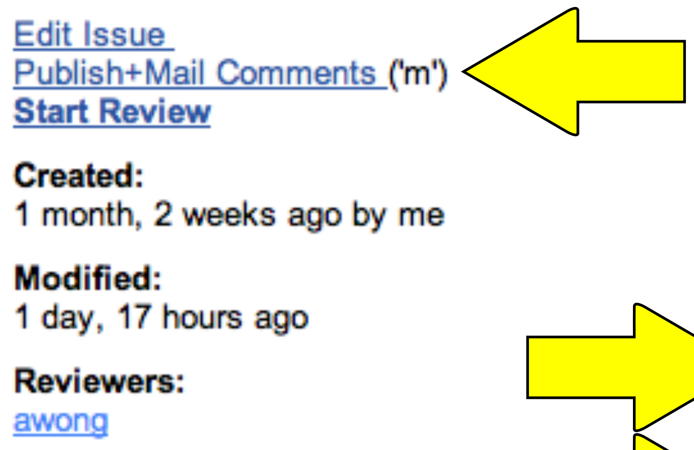
If you're feeling crafty you can also use one-liners:

```
git log --format=format:"%an" [path] | \
    sort | uniq -c | sort -n
```

It's your responsibility to find qualified reviewers for your change!

Sending out review

No one will notice your code review until you **Publish+Mail Comments** from reitveld.

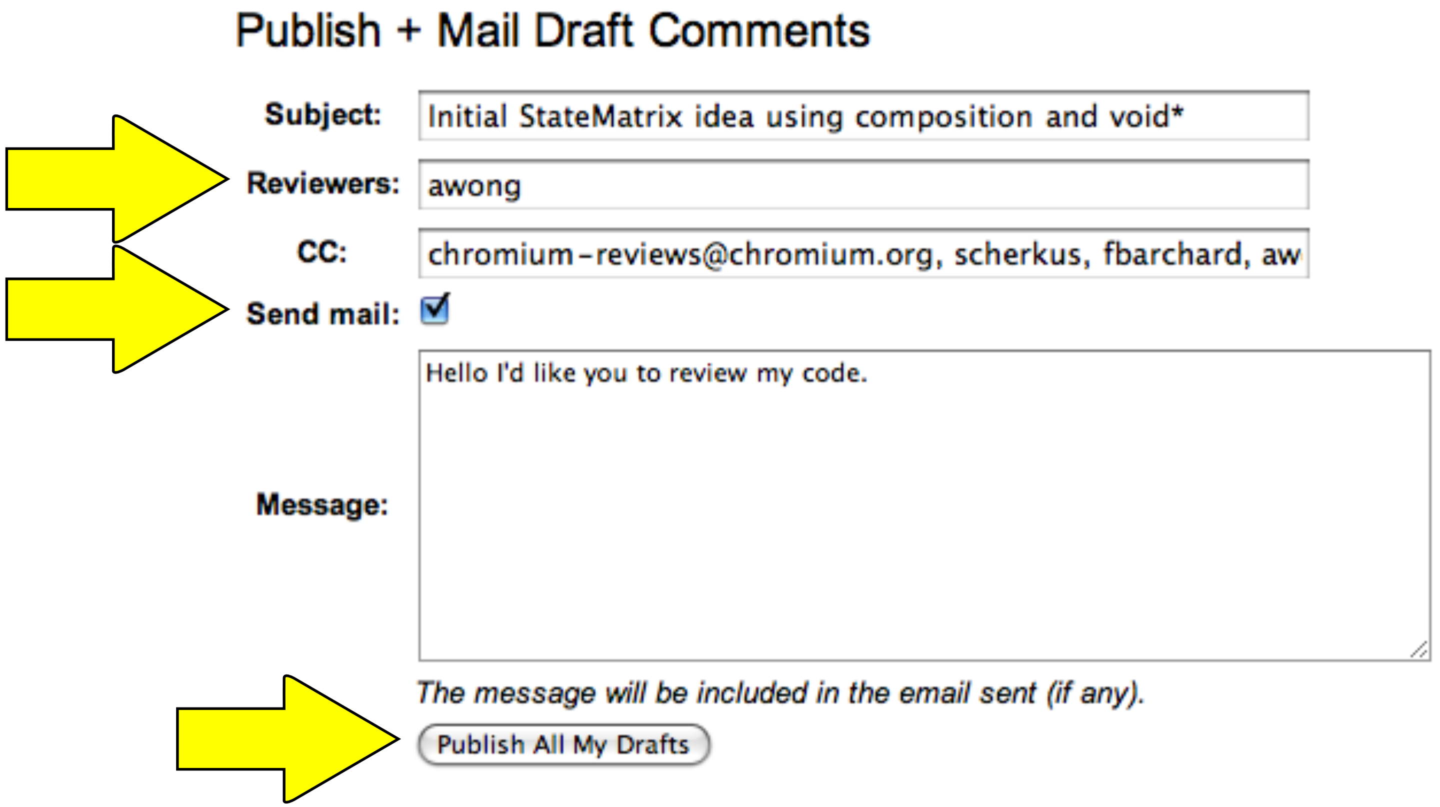


Edit Issue
Publish+Mail Comments ('m')
Start Review

Created:
1 month, 2 weeks ago by me

Modified:
1 day, 17 hours ago

Reviewers:
awong



Publish + Mail Draft Comments

Subject:	Initial StateMatrix idea using composition and void*
Reviewers:	awong
CC:	chromium-reviews@chromium.org, scherkus, fbarchard, aw
Send mail:	✓
Message:	Hello I'd like you to review my code.

The message will be included in the email sent (if any).

Sending out review

Reviewers not responding? It's very likely their email client filtered away the code review email.

Feel free to try the following:

- Using **Publish+Mail Comments** again to "ping" the reviewers (highly recommended)
- Emailing the reviewers directly
- Asking for a reviewer on Chromium IRC channel
- Asking for a reviewer on chromium-dev mailing list

In general, feel free to ping reviewers via **Publish+Mail Comments** if they have failed to respond within 24-48 hours.

Tips

Follow the style guide!!!

Write a descriptive, easy to understand, change description.

Always try to include tests when possible.

Keep your changes small. It's much easier for reviewers to understand and review your change. Split up unrelated changes.

Before attempting a big change, email your reviewers and discuss to make sure your approach is good.

Read contributing code for more information.

Guidelines

- Do the right thing for the project, not the fastest thing to get code checked in.
- If you touch code you're not familiar with (e.g. IPC, TabContents, testing infrastructure), add people who know that code as reviewers, even if the change seems simple.
- Similarly, if you're asked to review code you're not familiar with, add better reviewers, even if the change seems simple.
- Never commit a CL if there are unresolved comments from one of the reviewers, even if others said "LGTM".

6. Committing code

Committing code

If you **don't** have commit access, one of your reviewers should land your change for you.

If you **do** have commit access, first check the tree...

Red? Wait until tree goes green.

Green? Before committing make sure...

- ...your most recent try server attempt has passed
- ...you're in IRC or generally available on IM
- ...you're not leaving to eat lunch or catch a bus

Committing code

All good? OK let's commit!
Subversion: gcl commit my_new_feature
Git: checkout appropriate branch and run git cl dcommit

Make sure to keep watching the tree to make sure your change successfully builds without breaking anything.

Don't commit and leave!

Getting Commit Access

Like most open source projects, Chromium has rules.

Googlers don't get a free pass into Chromium land and typically must write a few patches before getting access.

Basic process:

1. Check out read-only version of the code
2. Write and land some patches
3. Get provisional committer access
4. Check out read/write version of the code
5. Write and land an additional ~20 patches
6. Get nominated for full committer status

Getting Commit Access

Q: Why can't I be a provisional committer right away?

A: It's best to have an experienced committer review and land your first few patches so you don't break the build.

You don't want to break the build, do you?

Troubleshooting

In open source, no one can hear you scream

IRC? freenode.net? #chromium?!?

If you weren't using the internet in the 80s and early 90s (before IM), you probably don't remember IRC.

IRC is a federated chat system. You log into a set of servers, pick a nick, and join a channel.

For chromium, the development channel is #chromium on freenode.net.

Discussion on #chromium is for facilitating development. Ask questions about build failures, tool issues, who owns what component, etc.

IRC is your friend. Learn to use it.

Feeling lost? Need help?

First, do your research:

- Search mail archives & change history
- If applicable, see what older revisions, or other browsers do

Then:

- Ask on IRC
- E-mail chromium-dev

For best results, include enough information for someone to understand your question. This includes:

- Build platform
- What revision/version you are working with
- Describe the expected behavior
- Describe what you've tried, and prior research

THE END

Start writing code!