# tl;dr

Summary in command line form

# Life of a Chromium Developer

# Git Workflow

Get into work...

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

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

```
$ git checkout -t -b my_new_feature origin/master
$ git cl upload
$ git cl 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 cl try
```

# Committing the patch

## Use the commit queue (CQ)

It’s usually a “commit” button or checkbox on code review.

If it’s not available for your project, or there are issues, report a bug and use Infra>Platform>CQ component. Leave the bug Untriaged so it can be reviewed during regular CQ bug triage.

If you are a committer and CQ has issues, feel free to land the patch manually.

# 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

**Infrastructure issues**
https://chromium.googlesource.com/infra/infra/+/master/doc/users/contacting_troopers.md

# Overview

Lots of information ahead!

# Overview

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

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!

- ~20000 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
- By platform: Windows, MacOS, Linux.

Google Employees:

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

# 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 2 steps:

1. Install Depot Tools (a collection of dev utilities)
2. Run `fetch chromium` 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

# 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):

```
$ git clone https://chromium.googlesource.com/chromium/tools/depot_tools.git
$ 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 Gerrit for review

Common utilities:

- gclient - syncs your source tree and creates build files.
- git-cl - integration with code review and tryjobs

# Depot Tools Quirks

Common Quirks:

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

# 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

```
git clone https://chromium.googlesource.com/chromium/tools/depot_tools.gitexport
PATH=`pwd`/depot_tools:"$PATH" # or windows equiv
```

2. Download the code

```
fetch chromium
```

# 3. Modifying and Building

# Development Environment (IDE, etc.)

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

Common Setups:

- Linux: vim/emacs
- Mac: Xcode
- Windows: Visual Studio

For code completion: Chromium Code Search :)

# Building Code: ninja

This should work on any OS (Linux, Mac, Windows)

In the **`src/`** directory:

```
$ gn gen out/Default
```

Set build arguments (such as debug vs release):

```
$ gn args out/Default
```

Do a build of the ‘chrome’ target:

```
$ autoninja -C out/Default chrome
# ‘autoninja’ is preferred over ‘ninja’
```

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

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

# Common Build Problems

**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
- Browser tests
- Performance tests
- Web tests

  (note: to build tests locally, you need a different build target other than ‘chrome’. Check out BUILD.gn files to find the correct target name)

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.

Web tests are written in HTML/CSS/Javascript and are hosted in Blink’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 teammates 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 uploaded change, applies it to a clean checkout of the source, and compiles and runs tests affected by the patch.

To submit your change to the try server:

Git: checkout branch you want to try and run `git cl try`

**Requires commit access!**

**See How To Become A Committer for details.**

# Try Server

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

- 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.

# 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 Gerrit instance at https://chromium-review.googlesource.com/

**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 Gerrit and log in with your chromium.org account or a Google account of your choosing.

Upload your change using the following:

Git: checkout appropriate branch and run `git cl upload`

If prompted, enter the same credentials you used to log into Gerrit above.

# Uploading code

Write a **meaningful and descriptive changelist description** and fill in the BUG= field. If a bug doesn't exist, go ahead and file one just for your change.

View your uploaded change in Gerrit (URL should be printed to console after running `git cl upload`).

# Finding reviewers

Best to ask your teammates or inspect commit logs:

```
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 **Start Review** in Gerrit (Note: These images reference Rietveld instead of Gerrit, and are out of date).

# 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 **Reply** 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 **Reply** 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.
- Don’t commit a CL if there are unresolved comments from one of the reviewers, even if others said "LGTM".

# 6. Committing code

# Committing the patch

## Use the commit queue (CQ)

It’s usually a “commit” button or checkbox on code review.

If it’s not available for your project, or there are issues, report a bug and use Build-CommitQueue label. Leave the bug Untriaged so it can be reviewed during regular CQ bug triage.

If you are a committer and CQ has issues, feel free to land the patch manually.

# Manual 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!

```
(chromium)$ git cl land      (blink) $ git cl dcommit
```

**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 nominated for full committer status

# Troubleshooting

In open source, no one can hear you scream

# Slack? IRC?

Some Chromium developers communicate on Slack or IRC.
Lately Slack has been more active.

See instructions at Slack and IRC.

# 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 Slack (or 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!