Checking out and building Chromium on Linux

There are instructions for other platforms linked from the get the code page.

Instructions for Google Employees

Are you a Google employee? See go/building-chrome instead.

Contents

System requirements

- A 64-bit Intel machine with at least 8GB of RAM. More than 16GB is highly recommended.
- At least 100GB of free disk space.
- You must have Git and Python v2 installed already.

Most development is done on Ubuntu (currently 16.04, Xenial Xerus). There are some instructions for other distros below, but they are mostly unsupported.

Install `depot_tools`

Clone the `depot_tools` repository:

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

Add `depot_tools` to the end of your PATH (you will probably want to put this in your `~/.bashrc` or `~/.zshrc`). Assuming you cloned `depot_tools` to `/path/to/depot_tools`:

```
$ export PATH="$PATH:/path/to/depot_tools"
```

When cloning `depot_tools` to your home directory **do not** use `~` on PATH, otherwise `gclient runhooks` will fail to run. Rather, you should use either `$HOME` or the absolute path:

```
$ export PATH="$PATH:${HOME}/depot_tools"
```

Get the code

Create a `chromium` directory for the checkout and change to it (you can call this whatever you like and put it wherever you like, as long as the full path has no spaces):

```
$ mkdir ~/chromium && cd ~/chromium
```

Run the `fetch` tool from depot_tools to check out the code and its dependencies.

```
$ fetch --nohooks chromium
```

If you don't want the full repo history, you can save a lot of time by adding the `--no-history` flag to `fetch`.

Expect the command to take 30 minutes on even a fast connection, and many hours on slower ones.

If you've already installed the build dependencies on the machine (from another checkout, for example), you can omit the `--nohooks` flag and `fetch` will automatically execute `gclient runhooks` at the end.

When `fetch` completes, it will have created a hidden `.gclient` file and a directory called `src` in the working directory. The remaining instructions assume you have switched to the `src` directory:

```
$ cd src
```

Install additional build dependencies

Once you have checked out the code, and assuming you're using Ubuntu, run build/install-build-deps.sh

```
$ ./build/install-build-deps.sh
```

You may need to adjust the build dependencies for other distros. There are some notes at the end of this document, but we make no guarantees for their accuracy.

Run the hooks

Once you've run `install-build-deps` at least once, you can now run the Chromium-specific hooks, which will download additional binaries and other things you might need:

```
$ gclient runhooks
```

Optional: You can also install API keys if you want your build to talk to some Google services, but this is not necessary for most development and testing purposes.

Setting up the build

Chromium uses Ninja as its main build tool along with a tool called GN to generate `.ninja` files. You can create any number of *build directories* with different configurations. To create a build directory, run:

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

- You only have to run this once for each new build directory, Ninja will update the build files as needed.
- You can replace `Default` with another name, but it should be a subdirectory of `out`.
- For other build arguments, including release settings, see GN build configuration. The default will be a debug component build matching the current host operating system and CPU.
- For more info on GN, run `gn help` on the command line or read the quick start guide.

Faster builds

This section contains some things you can change to speed up your builds, sorted so that the things that make the biggest difference are first.

Use Goma

Google developed the distributed compiler called Goma. Googlers and contributors who have tryjob access could use `Goma`.

If you are not a Googler and would like to use `Goma` sign up.

Once you're allowed to use `Goma` and have installed the client, set the following GN args:

```
use_goma=true
goma_dir="/path/to/goma-client"
```

Disable NaCl

By default, the build includes support for Native Client (NaCl), but most of the time you won't need it. You can set the GN argument `enable_nacl=false` and it won't be built.

Include fewer debug symbols

By default GN produces a build with all of the debug assertions enabled (`is_debug=true`) and including full debug info (`symbol_level=2`). Setting `symbol_level=1` will produce enough information for stack traces, but not line-by-line debugging. Setting `symbol_level=0` will include no debug symbols at all. Either will speed up the build compared to full symbols.

Disable debug symbols for Blink

Due to its extensive use of templates, the Blink code produces about half of our debug symbols. If you don't ever need to debug Blink, you can set the GN arg `blink_symbol_level=0`.

Use Icecc

Icecc is the distributed compiler with a central scheduler to share build load. Currently, many external contributors use it. e.g. Intel, Opera, Samsung (this is not useful if you're using Goma).

In order to use `icecc`, set the following GN args:

```
use_debug_fission=false
is_clang=false
```

See these links for more on the bundled_binutils limitation, the debug fission limitation.

Using the system linker may also be necessary when using glibc 2.21 or newer. See related bug.

ccache

You can use ccache to speed up local builds (again, this is not useful if you're using Goma).

Increase your ccache hit rate by setting `CCACHE_BASEDIR` to a parent directory that the working directories all have in common (e.g., `/home/yourusername/development`). Consider using `CCACHE_SLOPPINESS=include_file_mtime` (since if you are using multiple working directories, header times in svn sync'ed portions of your trees will be different - see the ccache troubleshooting section for additional information). If you use symbolic links from your home directory to get to the local physical disk directory where you keep those working development directories, consider putting

```
alias cd="cd -P"
```

in your `.bashrc` so that `$PWD` or `cwd` always refers to a physical, not logical directory (and make sure `CCACHE_BASEDIR` also refers to a physical parent).

If you tune ccache correctly, a second working directory that uses a branch tracking trunk and is up to date with trunk and was gclient sync'ed at about the same time should build chrome in about 1/3 the time, and the cache misses as reported by `ccache -s` should barely increase.

This is especially useful if you use `git-new-workdir` and keep multiple local working directories going at once.

Using tmpfs

You can use tmpfs for the build output to reduce the amount of disk writes required. I.e. mount tmpfs to the output directory where the build output goes:

As root:

```
mount -t tmpfs -o size=20G,nr_inodes=40k,mode=1777 tmpfs /path/to/out
```

Caveat: You need to have enough RAM + swap to back the tmpfs. For a full debug build, you will need about 20 GB. Less for just building the chrome target or for a release build.

Quick and dirty benchmark numbers on a HP Z600 (Intel core i7, 16 cores hyperthreaded, 12 GB RAM)

- With tmpfs:
  - 12m:20s
- Without tmpfs
  - 15m:40s

Build Chromium

Build Chromium (the "chrome" target) with Ninja using the command:

```
$ autoninja -C out/Default chrome
```

(`autoninja` is a wrapper that automatically provides optimal values for the arguments passed to `ninja`.)

You can get a list of all of the other build targets from GN by running `gn ls out/Default` from the command line. To compile one, pass the GN label to Ninja with no preceding "//" (so, for `//chrome/test:unit_tests` use `autoninja -C out/Default chrome/test:unit_tests`).

Run Chromium

Once it is built, you can simply run the browser:

```
$ out/Default/chrome
```

Running test targets

You can run the tests in the same way. You can also limit which tests are run using the `--gtest_filter` arg, e.g.:

```
$ out/Default/unit_tests --gtest_filter="PushClientTest.*"
```

You can find out more about GoogleTest at its GitHub page.

Update your checkout

To update an existing checkout, you can run

```
$ git rebase-update
$ gclient sync
```

The first command updates the primary Chromium source repository and rebases any of your local branches on top of tip-of-tree (aka the Git branch `origin/master`). If you don't want to use this script, you can also just use `git pull` or other common Git commands to update the repo.

The second command syncs dependencies to the appropriate versions and re-runs hooks as needed.

Tips, tricks, and troubleshooting

Linker Crashes

If, during the final link stage:

```
LINK out/Debug/chrome
```

You get an error like:

```
collect2: ld terminated with signal 6 Aborted terminate called after throwing an instance of 's
collect2: ld terminated with signal 11 [Segmentation fault], core dumped
```

you are probably running out of memory when linking. You *must* use a 64-bit system to build. Try the following build settings (see GN build configuration for other settings):

- Build in release mode (debugging symbols require more memory): `is_debug = false`
- Turn off symbols: `symbol_level = 0`
- Build in component mode (this is for development only, it will be slower and may have broken functionality): `is_component_build = true`

More links

- Information about building with Clang.
- You may want to use a chroot to isolate yourself from versioning or packaging conflicts.
- Cross-compiling for ARM? See LinuxChromiumArm.
- Want to use Eclipse as your IDE? See LinuxEclipseDev.
- Want to use your built version as your default browser? See LinuxDevBuildAsDefaultBrowser.

Next Steps

If you want to contribute to the effort toward a Chromium-based browser for Linux, please check out the Linux Development page for more information.

Notes for other distros

Arch Linux

Instead of running `install-build-deps.sh` to install build dependencies, run:

```
$ sudo pacman -S --needed python perl gcc gcc-libs bison flex gperf pkgconfig \
nss alsa-lib glib2 gtk3 nspr ttf-ms-fonts freetype2 cairo dbus libgnome-keyring
```

For the optional packages on Arch Linux:

- `php-cgi` is provided with `pacman`
- `wdiff` is not in the main repository but `dwdiff` is. You can get `wdiff` in AUR/`yaourt`
- `sun-java6-fonts` do not seem to be in main repository or AUR.

Crostini (Debian based)

First install the `file` and `lsb-release` commands for the script to run properly:

```
$ sudo apt-get install file lsb-release
```

Then invoke install-build-deps.sh with the `--no-arm` argument, because the ARM toolchain doesn't exist for this configuration:

```
$ sudo install-build-deps.sh --no-arm
```

Fedora

Instead of running `build/install-build-deps.sh`, run:

```
su -c 'yum install git python bzip2 tar pkgconfig atk-devel alsa-lib-devel \
bison binutils brlapi-devel bluez-libs-devel bzip2-devel cairo-devel \
cups-devel dbus-devel dbus-glib-devel expat-devel fontconfig-devel \
freetype-devel gcc-c++ glib2-devel glibc.i686 gperf glib2-devel \
gtk3-devel java-1.*.0-openjdk-devel libatomic libcap-devel libffi-devel \
libgcc.i686 libgnome-keyring-devel libjpeg-devel libstdc++.i686 libX11-devel \
libXScrnSaver-devel libXtst-devel libxkbcommon-x11-devel ncurses-compat-libs \
nspr-devel nss-devel pam-devel pango-devel pciutils-devel \
pulseaudio-libs-devel zlib.i686 httpd mod_ssl php php-cli python-psutil wdiff \
xorg-x11-server-Xvfb'
```

The fonts needed by Blink's web tests can be obtained by following these instructions. For the optional packages:

- `php-cgi` is provided by the `php-cli` package.
- `sun-java6-fonts` is covered by the instructions linked above.

Gentoo

You can just run `emerge www-client/chromium`.

OpenSUSE

Use `zypper` command to install dependencies:

(openSUSE 11.1 and higher)

```
sudo zypper in subversion pkg-config python perl bison flex gperf \
     mozilla-nss-devel glib2-devel gtk-devel wdiff lighttpd gcc gcc-c++ \
     mozilla-nspr mozilla-nspr-devel php5-fastcgi alsa-devel libexpat-devel \
     libjpeg-devel libbz2-devel
```

For 11.0, use `libnspr4-0d` and `libnspr4-dev` instead of `mozilla-nspr` and `mozilla-nspr-devel`, and use `php5-cgi` instead of `php5-fastcgi`.

(openSUSE 11.0)

```
sudo zypper in subversion pkg-config python perl \
     bison flex gperf mozilla-nss-devel glib2-devel gtk-devel \
     libnspr4-0d libnspr4-dev wdiff lighttpd gcc gcc-c++ libexpat-devel \
     php5-cgi alsa-devel gtk3-devel jpeg-devel
```

The Ubuntu package `sun-java6-fonts` contains a subset of Java of the fonts used. Since this package requires Java as a prerequisite anyway, we can do the same thing by just installing the equivalent openSUSE Sun Java package:

```
sudo zypper in java-1_6_0-sun
```

WebKit is currently hard-linked to the Microsoft fonts. To install these using `zypper`

```
sudo zypper in fetchmsttfonts pullin-msttf-fonts
```

To make the fonts installed above work, as the paths are hardcoded for Ubuntu, create symlinks to the appropriate locations:

```
sudo mkdir -p /usr/share/fonts/truetype/msttcorefonts
sudo ln -s /usr/share/fonts/truetype/arial.ttf /usr/share/fonts/truetype/msttcorefonts/Arial.tt
sudo ln -s /usr/share/fonts/truetype/arialbd.ttf /usr/share/fonts/truetype/msttcorefonts/Arial_
sudo ln -s /usr/share/fonts/truetype/arialbi.ttf /usr/share/fonts/truetype/msttcorefonts/Arial_
sudo ln -s /usr/share/fonts/truetype/ariali.ttf /usr/share/fonts/truetype/msttcorefonts/Arial_I
sudo ln -s /usr/share/fonts/truetype/comic.ttf /usr/share/fonts/truetype/msttcorefonts/Comic_Sa
sudo ln -s /usr/share/fonts/truetype/comicbd.ttf /usr/share/fonts/truetype/msttcorefonts/Comic_
sudo ln -s /usr/share/fonts/truetype/cour.ttf /usr/share/fonts/truetype/msttcorefonts/Courier_N
sudo ln -s /usr/share/fonts/truetype/courbd.ttf /usr/share/fonts/truetype/msttcorefonts/Courier
sudo ln -s /usr/share/fonts/truetype/courbi.ttf /usr/share/fonts/truetype/msttcorefonts/Courier
sudo ln -s /usr/share/fonts/truetype/couri.ttf /usr/share/fonts/truetype/msttcorefonts/Courier_
sudo ln -s /usr/share/fonts/truetype/impact.ttf /usr/share/fonts/truetype/msttcorefonts/Impact.
sudo ln -s /usr/share/fonts/truetype/times.ttf /usr/share/fonts/truetype/msttcorefonts/Times_Ne
sudo ln -s /usr/share/fonts/truetype/timesbd.ttf /usr/share/fonts/truetype/msttcorefonts/Times_
sudo ln -s /usr/share/fonts/truetype/timesbi.ttf /usr/share/fonts/truetype/msttcorefonts/Times_
sudo ln -s /usr/share/fonts/truetype/timesi.ttf /usr/share/fonts/truetype/msttcorefonts/Times_N
sudo ln -s /usr/share/fonts/truetype/verdana.ttf /usr/share/fonts/truetype/msttcorefonts/Verdan
sudo ln -s /usr/share/fonts/truetype/verdanab.ttf /usr/share/fonts/truetype/msttcorefonts/Verda
sudo ln -s /usr/share/fonts/truetype/verdanai.ttf /usr/share/fonts/truetype/msttcorefonts/Verda
sudo ln -s /usr/share/fonts/truetype/verdanaz.ttf /usr/share/fonts/truetype/msttcorefonts/Verda
```

The Ubuntu package `sun-java6-fonts` contains a subset of Java of the fonts used. Since this package requires Java as a prerequisite anyway, we can do the same thing by just installing the equivalent openSUSE Sun Java package:

```
sudo zypper in java-1_6_0-sun
```

WebKit is currently hard-linked to the Microsoft fonts. To install these using `zypper`

```
sudo zypper in fetchmsttfonts pullin-msttf-fonts
```

To make the fonts installed above work, as the paths are hardcoded for Ubuntu, create symlinks to the appropriate locations:

```
sudo mkdir -p /usr/share/fonts/truetype/msttcorefonts
sudo ln -s /usr/share/fonts/truetype/arial.ttf /usr/share/fonts/truetype/msttcorefonts/Arial.tt
sudo ln -s /usr/share/fonts/truetype/arialbd.ttf /usr/share/fonts/truetype/msttcorefonts/Arial_
sudo ln -s /usr/share/fonts/truetype/arialbi.ttf /usr/share/fonts/truetype/msttcorefonts/Arial_
sudo ln -s /usr/share/fonts/truetype/ariali.ttf /usr/share/fonts/truetype/msttcorefonts/Arial_I
sudo ln -s /usr/share/fonts/truetype/comic.ttf /usr/share/fonts/truetype/msttcorefonts/Comic_Sa
sudo ln -s /usr/share/fonts/truetype/comicbd.ttf /usr/share/fonts/truetype/msttcorefonts/Comic_
sudo ln -s /usr/share/fonts/truetype/cour.ttf /usr/share/fonts/truetype/msttcorefonts/Courier_N
sudo ln -s /usr/share/fonts/truetype/courbd.ttf /usr/share/fonts/truetype/msttcorefonts/Courier
sudo ln -s /usr/share/fonts/truetype/courbi.ttf /usr/share/fonts/truetype/msttcorefonts/Courier
sudo ln -s /usr/share/fonts/truetype/couri.ttf /usr/share/fonts/truetype/msttcorefonts/Courier_
sudo ln -s /usr/share/fonts/truetype/impact.ttf /usr/share/fonts/truetype/msttcorefonts/Impact.
sudo ln -s /usr/share/fonts/truetype/times.ttf /usr/share/fonts/truetype/msttcorefonts/Times_New
sudo ln -s /usr/share/fonts/truetype/timesbd.ttf /usr/share/fonts/truetype/msttcorefonts/Times_
sudo ln -s /usr/share/fonts/truetype/timesbi.ttf /usr/share/fonts/truetype/msttcorefonts/Times_
sudo ln -s /usr/share/fonts/truetype/timesi.ttf /usr/share/fonts/truetype/msttcorefonts/Times_N
sudo ln -s /usr/share/fonts/truetype/verdana.ttf /usr/share/fonts/truetype/msttcorefonts/Verdana
sudo ln -s /usr/share/fonts/truetype/verdanab.ttf /usr/share/fonts/truetype/msttcorefonts/Verda
sudo ln -s /usr/share/fonts/truetype/verdanai.ttf /usr/share/fonts/truetype/msttcorefonts/Verda
sudo ln -s /usr/share/fonts/truetype/verdanaz.ttf /usr/share/fonts/truetype/msttcorefonts/Verda
```

And then for the Java fonts:

```
sudo mkdir -p /usr/share/fonts/truetype/ttf-lucida
sudo find /usr/lib*/jvm/java-1.6.*-sun-*/jre/lib -iname '*.ttf' -print \
     -exec ln -s {} /usr/share/fonts/truetype/ttf-lucida \;
```