Chapter 8. Customizing Git

Git Configuration

- One of the first things you did was set up your name and e-mail address:

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
```

- The first place Git looks for these values is in an `/etc/gitconfig` file, which contains values for every user on the system and all of their repositories. If you pass the option `\\--system` to `git config`, it reads and writes from this file specifically.
- The next place Git looks is the `~/.gitconfig` (or `~/.config/git/config`) file, which is specific to each user. You can make Git read and write to this file by passing the `\\--global` option.
- Finally, Git looks for configuration values in the configuration file in the Git directory (`.git/config`) of whatever repository youre currently using. These values are specific to that single repository.

Basic Client Configuration

- The configuration options recognized by Git fall into two categories: client- side and server-side.
- The majority of the options are client-side - configuring your personal working preferences.

```
$ man git-config
```

`core.editor`

- By default, Git uses whatever youve set as your default text editor (`$VISUAL` or `$EDITOR`) or else falls back to the `vi` editor to create and edit your commit and tag messages.

```
$ git config --global core.editor emacs
```

commit.template

- If you set this to the path of a file on your system, Git will use that file as the default message when you commit.
- For instance, suppose you create a template file at `~/.gitmessage.txt` that looks like this:

```
subject line

what happened

[ticket: X]
```

- To tell Git to use it as the default message that appears in your editor when you run `git commit`, set the `commit.template` configuration value:

```
$ git config --global commit.template ~/.gitmessage.txt
$ git commit
```

- Then, your editor will open to something like this for your placeholder commit message when you commit:

```
subject line

what happened

[ticket: X]
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:   lib/test.rb
#
~
~
".git/COMMIT_EDITMSG" 14L, 297C
```

core.pager

- This setting determines which pager is used when Git pages output such as `log` and `diff`.
- You can set it to `more` or to your favorite pager (by default, its `less`), or you can turn it off by setting it to a blank string:

```
$ git config --global core.pager ''
```

user.signingkey

- If youre making signed annotated tags, setting your GPG signing key as a configuration setting makes things easier.
- Set your key ID like so:

```
$ git config --global user.signingkey <gpg-key-id>
```

- Now, you can sign tags without having to specify your key every time with the `git tag` command:

```
$ git tag -s <tag-name>
```

core.excludesfile

- Sometimes you want to ignore certain files for all repositories that you work with.
- If your computer is running Mac OS X, youre probably familiar with `.DS_Store` files.
- If your preferred editor is Emacs or Vim, you know about files that end with a `~`.
- This setting lets you write a kind of global `.gitignore` file. If you create a `~/.gitignore_global` file with these contents:

```
*~
.DS_Store
```

help.autocorrect

- If you mistype a command, it shows you something like this:

```
$ git chekcout master
git: 'chekcout' is not a git command. See 'git --help'.

Did you mean this?
    checkout
```

- Git helpfully tries to figure out what you meant, but it still refuses to do it.
- If you set `help.autocorrect` to 1, Git will actually run this command for you:

```
$ git chekcout master
WARNING: You called a Git command named 'chekcout', which does not exist.
Continuing under the assumption that you meant 'checkout'
in 0.1 seconds automatically...
```

Colors in Git

`color.ui`

- To turn off all Gits colored terminal output, do this:

```
$ git config --global color.ui false
```

- You can also set it to `always` to ignore the difference between terminals and pipes.

`color.*`

- If you want to be more specific about which commands are colored and how, Git provides verb-specific coloring settings.
- Each of these can be set to `true`, `false`, or `always`:

```
color.branch
color.diff
color.interactive
color.status
```

- For example, to set the meta information in your diff output to blue foreground, black background, and bold text, you can run

```
$ git config --global color.diff.meta "blue black bold"
```

- You can set the color to any of the following values: `normal`, `black`, `red`, `green`, `yellow`, `blue`, `magenta`, `cyan`, or `white`.
- If you want an attribute like bold, you can choose from `bold`, `dim`, `ul` (underline), `blink`, and `reverse` (swap foreground and background).

External Merge and Diff Tools

- P4Merge works on all major platforms, so you should be able to do so.
- Well use path names in the examples that work on Mac and Linux systems; for Windows, youll have to change `/usr/local/bin` to an executable path in your environment.

```
$ cat /usr/local/bin/extMerge
#!/bin/sh
/Applications/p4merge.app/Contents/MacOS/p4merge $*
```

- The diff wrapper checks to make sure seven arguments are provided and passes two of them to your merge script.
- By default, Git passes the following arguments to the diff program:

```
path old-file old-hex old-mode new-file new-hex new-mode
```

- Because you only want the `old-file` and `new-file` arguments, you use the wrapper script to pass the ones you need.

```
$ cat /usr/local/bin/extDiff
#!/bin/sh
[ $# -eq 7 ] && /usr/local/bin/extMerge "$2" "$5"
```

- You also need to make sure these tools are executable:

```
$ sudo chmod +x /usr/local/bin/extMerge
$ sudo chmod +x /usr/local/bin/extDiff
```

- Now you can set up your config file to use your custom merge resolution and diff tools.
- This takes a number of custom settings: `merge.tool` to tell Git what strategy to use, `mergetool.<tool>.cmd` to specify how to run the command, `mergetool.<tool>.trustExitCode` to tell Git if the exit code of that program indicates a successful merge resolution or not, and `diff.external` to tell Git what command to run for diffs.

```
$ git config --global merge.tool extMerge
$ git config --global mergetool.extMerge.cmd \
  'extMerge \"$BASE\" \"$LOCAL\" \"$REMOTE\" \"$MERGED\"'
$ git config --global mergetool.extMerge.trustExitCode false
$ git config --global diff.external extDiff
```

- or you can edit your `~/.gitconfig` file to add these lines:

```
[merge]
  tool = extMerge
[mergetool "extMerge"]
  cmd = extMerge "$BASE" "$LOCAL" "$REMOTE" "$MERGED"
  trustExitCode = false
[diff]
  external = extDiff
```

- After all this is set, if you run diff commands such as this:

```
$ git diff 32d1776b1^ 32d1776b1
```

- Instead of getting the diff output on the command line, Git fires up P4Merge, which looks something like this:

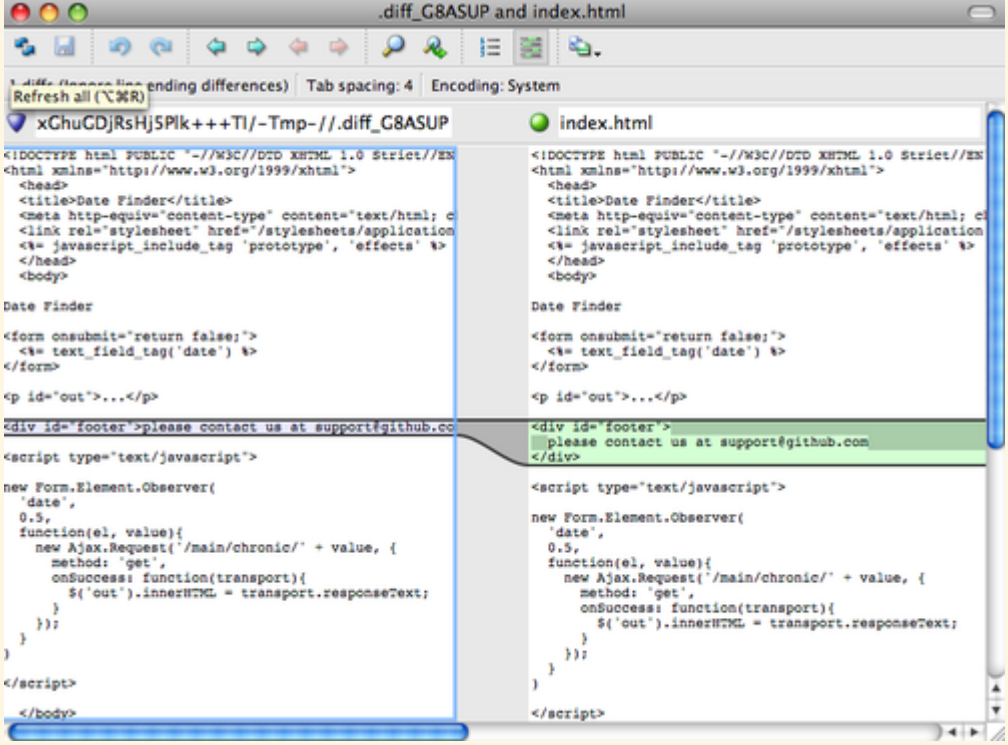

Figure 8-1. P4Merge.

- For example, to change your `extDiff` and `extMerge` tools to run the KDiff3 tool instead, all you have to do is edit your `extMerge` file:

```
$ cat /usr/local/bin/extMerge
#!/bin/sh
/Applications/kdiff3.app/Contents/MacOS/kdiff3 $*
```

- To see a list of the tools it supports, try this:

```
$ git mergetool --tool-help
'git mergetool --tool=<tool>' may be set to one of the following:
        emerge
        gvimdiff
        gvimdiff2
        opendiff
        p4merge
        vimdiff
        vimdiff2

The following tools are valid, but not currently available:
        araxis
        bc3
        codecompare
        deltawalker
        diffmerge
        diffuse
        ecmerge
        kdiff3
        meld
        tkdiff
        tortoisemerge
        xxdiff
```

```
Some of the tools listed above only work in a windowed
environment. If run in a terminal-only session, they will fail.
```

- If youre not interested in using KDiff3 for diff but rather want to use it just for merge resolution, and the kdiff3 command is in your path, then you can run

```
$ git config --global merge.tool kdiff3
```

Formatting and Whitespace

`core.autocrlf`

- Windows uses both a carriage-return character and a linefeed character for newlines in its files, whereas Mac and Linux systems use only the linefeed character.
- This is a subtle but incredibly annoying fact of cross-platform work; many editors on Windows silently replace existing LF-style line endings with CRLF, or insert both line-ending characters when the user hits the enter key.
- Git can handle this by auto-converting CRLF line endings into LF when you add a file to the index, and vice versa when it checks out code onto your filesystem.
- You can turn on this functionality with the `core.autocrlf` setting.
- If youre on a Windows machine, set it to `true` - this converts LF endings into CRLF when you check out code:

```
$ git config --global core.autocrlf true
```

- If youre on a Linux or Mac system that uses LF line endings, then you dont want Git to automatically convert them when you check out files; however, if a file with CRLF endings accidentally gets introduced, then you may want Git to fix it.
- You can tell Git to convert CRLF to LF on commit but not the other way around by setting `core.autocrlf` to input:

```
$ git config --global core.autocrlf input
```

- If youre a Windows programmer doing a Windows-only project, then you can turn off this functionality, recording the carriage returns in the repository by setting the config value to `false`:

```
$ git config --global core.autocrlf false
```

`core.whitespace`

- The ones that are turned on by default are `blank-at-eol`, which looks for spaces at the end of a line; `blank-at-eof`, which notices blank lines at the end of a file; and `space-before-tab`, which looks for spaces before tabs at the beginning of a line.

- The three that are disabled by default but can be turned on are `indent-with- non-tab`, which looks for lines that begin with spaces instead of tabs (and is controlled by the `tabwidth` option); `tab-in-indent`, which watches for tabs in the indentation portion of a line; and `cr-at-eol`, which tells Git that carriage returns at the end of lines are OK.
- For example, if you want all but `cr-at-eol` to be set, you can do this:

```
$ git config --global core.whitespace \
    trailing-space,space-before-tab,indent-with-non-tab
```

- When youre applying patches, you can ask Git to warn you if its applying patches with the specified whitespace issues:

```
$ git apply --whitespace=warn <patch>
```

- Or you can have Git try to automatically fix the issue before applying the patch:

```
$ git apply --whitespace=fix <patch>
```

- These options apply to the `git rebase` command as well.
- If youve committed whitespace issues but havent yet pushed upstream, you can run `git rebase --whitespace=fix` to have Git automatically fix whitespace issues as its rewriting the patches.