Git diff

git add git commit git diff git stash .gitignore

Comparing changes with git diff

Diffing is a function that takes two input data sets and outputs the changes between them. `git diff` is a multi-use Git command that when executed runs a diff function on Git data sources. These data sources can be commits, branches, files and more. This document will discuss common invocations of `git diff` and diffing work flow patterns. The `git diff` command is often used along with `git status` and `git log` to analyze the current state of a Git repo.

Reading diffs: outputs

Raw output format

The following examples will be executed in a simple repo. The repo is created with the commands below:

```
$:> mkdir diff_test_repo
$:> cd diff_test_repo
$:> touch diff_test.txt
$:> echo "this is a git diff test example" > diff_test.txt
$:> git init .
Initialized empty Git repository in /Users/kev/code/test/.git/
$:> git add diff_test.txt
$:> git commit -am"add diff test file"
[main (root-commit) 6f77fc3] add diff test file
1 file changed, 1 insertion(+)
create mode 100644 diff_test.txt
```

If we execute `git diff` at this point, there will be no output. This is expected behavior as there are no changes in the repo to diff. Once the repo is created and we've added the `diff_test.txt` file, we can change the contents of the file to start experimenting with diff output.

```
$:> echo "this is a diff example" > diff_test.txt
```

Executing this command will change the content of the `diff_test.txt` file. Once modified, we can view a diff and analyze the output. Now executing `git diff` will produce the following output:

```
diff --git a/diff_test.txt b/diff_test.txt
index 6b0c6cf..b37e70a 100644
--- a/diff_test.txt
+++ b/diff_test.txt
@@ -1 +1 @@
```

```
-this is a git diff test example
+this is a diff example
```

Let us now examine a more detailed breakdown of the diff output.

1. Comparison input

```
diff --git a/diff_test.txt b/diff_test.txt
```

This line displays the input sources of the diff. We can see that `a/diff_test.txt` and `b/diff_test.txt` have been passed to the diff.

2. Meta data

```
index 6b0c6cf..b37e70a 100644
```

This line displays some internal Git metadata. You will most likely not need this information. The numbers in this output correspond to Git object version hash identifiers.

3. Markers for changes

```
--- a/diff_test.txt
+++ b/diff_test.txt
```

These lines are a legend that assigns symbols to each diff input source. In this case, changes from `a/diff_test.txt` are marked with a `---` and the changes from `b/diff_test.txt` are marked with the `+++` symbol.

4. Diff chunks

The remaining diff output is a list of diff 'chunks'. A diff only displays the sections of the file that have changes. In our current example, we only have one chunk as we are working with a simple scenario. Chunks have their own granular output semantics.

```
@@ -1 +1 @@
-this is a git diff test example
+this is a diff example
```

The first line is the chunk header. Each chunk is prepended by a header inclosed within `@@` symbols. The content of the header is a summary of changes made to the file. In our simplified example, we have -1 +1 meaning line one had changes. In a more realistic diff, you would see a header like:

```
@@ -34,6 +34,8 @@
```

In this header example, 6 lines have been extracted starting from line number 34. Additionally, 8 lines have been added starting at line number 34.

The remaining content of the diff chunk displays the recent changes. Each changed line is prepended with a `+` or `-` symbol indicating which version of the diff input the changes come from. As we previously discussed, `-` indicates changes from the `a/diff_test.txt` and `+` indicates changes from `b/diff_test.txt`.

Highlighting changes

1. `git diff --color-words`

`git diff` also has a special mode for highlighting changes with much better granularity: `--color-words`. This mode tokenizes added and removed lines by whitespace and then diffs those.

```
$:> git diff --color-words
diff --git a/diff_test.txt b/diff_test.txt
index 6b0c6cf..b37e70a 100644
--- a/diff_test.txt
+++ b/diff_test.txt
@@ -1 +1 @@
this is agit difftest example
```

Now the output displays only the color-coded words that have changed.

2. `git diff-highlight`

If you clone the git source, you'll find a sub-directory called contrib. It contains a bunch of git-related tools and other interesting bits and pieces that haven't yet been promoted to git core. One of these is a Perl script called diff-highlight. Diff-highlight pairs up matching lines of diff output and highlights sub-word fragments that have changed.

```
$:> git diff | /your/local/path/to/git-core/contrib/diff-highlight/diff-
highlight
diff --git a/diff_test.txt b/diff_test.txt
index 6b0c6cf..b37e70a 100644
--- a/diff_test.txt
+++ b/diff_test.txt
@@ -1 +1 @@
-this is a git diff test example
+this is a diff example
```

Now we've pared down our diff to the smallest possible change.

Diffing binary files

In addition to the text file utilities we have thus far demonstrated, `git diff` can be run on binary files. Unfortunately, the default output is not very helpful.

```
$:> git diff
Binary files a/script.pdf and b/script.pdf differ
```

Git does have a feature that allows you to specify a shell command to transform the content of your binary files into text prior to performing the diff. It does require a little set up though. First, you need to specify a textconv filter describing how to convert a certain type of binary to text. We're using a simple utility called pdftohtml (available via homebrew) to convert my PDFs into human readable HTML. You can set this up for a single repository by editing your `.git/config` file, or globally by editing `~ /.gitconfig`

```
[diff "pdfconv"]
textconv=pdftohtml -stdout
```

Then all you need to do is associate one or more file patterns with our pdfconv filter. You can do this by creating a `.gitattributes` file in the root of your repository.

```
*.pdf diff=pdfconv
```

Once configured, `git diff` will first run the binary file through the configured converter script and diff the converter output. The same technique can be applied to get useful diffs from all sorts of binary files, for example: zips, jars and other archives: using `unzip -l` (or similar) in place of pdf2html will show you paths that have been added or removed between commits images: exiv2 can be used to show metadata changes such as image dimensions documents: conversion tools exist for transforming .odf, .doc and other document formats to plain text. In a pinch, strings will often work for binary files where no formal converter exists.

Comparing files: git diff file

The `git diff` command can be passed an explicit file path option. When a file path is passed to `git diff` the diff operation will be scoped to the specified file. The below examples demonstrate this usage.

```
git diff HEAD ./path/to/file
```

This example is scoped to `./path/to/file` when invoked, it will compare the specific changes in the working directory, against the index, showing the changes that are not staged yet. By default `git diff` will execute the comparison against `HEAD`. Omitting `HEAD` in the example above `git diff ./path/to/file` has the same effect.

```
git diff --cached ./path/to/file
```

When `git diff` is invoked with the `--cached` option the diff will compare the staged changes with the local repository. The `--cached` option is synonymous with `--staged`.

Comparing all changes

Invoking `git diff` without a file path will compare changes across the entire repository. The above, file specific examples, can be invoked without the `./path/to/file` argument and have the same output results across all files in the local repo.

Changes since last commit

By default `git diff` will show you any uncommitted changes since the last commit.

```
git diff
```

Comparing files between two different commits

`git diff` can be passed Git refs to commits to diff. Some example refs are, `HEAD`, tags, and branch names. Every commit in Git has a commit ID which you can get when you execute `GIT LOG`. You can also pass this commit ID to `git diff`.

```
git log --pretty=oneline
957fbc92b123030c389bf8b4b874522bdf2db72c add feature
ce489262a1ee34340440e55a0b99ea6918e19e7a rename some classes
6b539f280d8b0ec4874671bae9c6bed80b788006 refactor some code for feature
646e7863348a427e1ed9163a9a96fa759112f102 add some copy to body

$:> git diff 957fbc92b123030c389bf8b4b874522bdf2db72c ce489262a1ee34340440e55
a0b99ea6918e19e7a
```

Comparing branches

Comparing two branches

Branches are compared like all other ref inputs to `git diff`

```
git diff branch1..other-feature-branch
```

This example introduces the dot operator. The two dots in this example indicate the diff input is the tips of both branches. The same effect happens if the dots are omitted and a space is used between the branches. Additionally, there is a three dot operator:

```
git diff branch1...other-feature-branch
```

The three dot operator initiates the diff by changing the first input parameter `branch1`. It changes `branch1` into a ref of the shared common ancestor commit between the two diff inputs, the shared ancestor of `branch1` and other-feature-branch. The last parameter input parameter remains unchanged as the tip of other-feature-branch.

Comparing files from two branches

To compare a specific file across branches, pass in the path of the file as the third argument to `git diff`

```
git diff main new_branch ./diff_test.txt
```

Summary

This page disscused the Git diffing process and the `git diff` command. We discussed how to read `git diff` output and the various data included in the output. Examples were provided on how to alter the `git diff` output with highlighting and colors. We discussed different diffing strategies such as how to diff files in branches and specific commits. In addition to the `git diff` command, we also used `git log` and `git checkout`.