Git: greatest hits

(also see `man gittutorial`, `man gitglossary`)

Querying

`git status`

What's going on in the repo right now?

Shows various pieces of information:

- Current branch
- Uncommitted changes
  - Untracked files
  - Tracked (*staged*) files
- Hints about commands you might use next

`--short (-s)`: Shows short-format listing of uncommitted changed files, one per line.

```
$ git status
On branch main
nothing to commit, working tree clean

$ touch new-file
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be
committed)

    new-file

nothing added to commit but untracked files present (use "git
add" to track)
$
```

What is the staging area (a.k.a. the index)?

A snapshot of the working tree indicating what will become part of the next commit you make.

- Only snapshots files with modifications
- Only snapshots files you've added to it with `git add`
- Can be updated at any time with `git add`, `git reset`, and others
- Not shared: each copy of a repository has its own index
- Resets every time you make a commit

Ways to refer to a commit

(See `man gitrevisions` for more detail.)

- **Hash** (e.g. `46e2f3b`): either the full SHA-1 hash, or enough of the beginning of the hash to match only a single object.
- **Ref** (e.g. `main`): generally a local or remote branch name. Looks in `.git/refs/` under the hood.
  - The special ref HEAD refers to the commit or branch that you're currently "on." HEAD is what commands like `git diff --cached` and `git status` compare against, and it's the parent of new commits you make.

When you need to specify a commit or branch, you can generally use either of these methods.

git log

What history is visible from the given commit (or HEAD) for the given files?

Shows all the parents of a given commit. Defaults to HEAD.

`-- <paths>`: Shows only commits for the given paths. `--` stops Git from treating a path as a branch.

`--patch (-p)`: Show the diff for each commit.

`--oneline`: Show each commit as a single line.

```
$ git log
commit e67e22ba38560e1a644d09e04afc4374bd5a2ebc (HEAD ->
main)
Author: Thomas Hebb <tommyhebb@gmail.com>
Date:   Wed Oct 6 08:35:19 2021 -0400

     Add a longer file in directory2

<...>

$ git log --oneline add-symlink
ff553ab (add-symlink) Add a symlink to file1
46e2f3b Initial commit

$ git log --oneline -- file2
8320c08 (HEAD -> main) Fix file2 to match file zoo
fe7b7f5 Add file2
$
```

`git show`

What's in this object (usually a commit)?

Shows commits, blobs, trees, tags in a human-readable format. Defaults to showing HEAD. Like `git log`, but doesn't show parents.

`--no-patch`: Don't show a commit's diff (same as `git log` without `-p`).

```
$ git show 46e2f3b
commit 46e2f3b72116845599bc868cb8aa65ddf23c5b9d
Author: Thomas Hebb <tommyhebb@gmail.com>
Date:   Tue Oct 5 18:20:02 2021 -0400

     Initial commit

     I like this file from the file zoo. Let's make a repo
with it!

diff --git a/file1 b/file1
new file mode 100644
index 0000000..a28a390
--- /dev/null
+++ b/file1
@@ -0,0 +1 @@
+I'm a file
$
```

`git diff`

What changed (since HEAD, since a given commit, or between two commits)?

By default, shows unstaged changes (i.e. the difference between the index and the working tree).

`--cached`: Shows the index compared to HEAD or the given commit.

`<commit>`: Shows the working tree compared to `<commit>`.

`<commit1> <commit2>`: Shows `<commit2>` compared to `<commit1>`.

```
$ echo 'A second line!' >>file1
$ git diff
diff --git a/file1 b/file1
index a28a390..6cec40f 100644
--- a/file1
+++ b/file1
@@ -1 +1,2 @@
 I'm a file
+A second line!
$
```

Operations on the working tree

`git add`

Update the index with a file from the working tree.

Takes a snapshot of the given file(s) and updates the index with that snapshot.

`--patch` (`-p`): Interactively select which changes within each file to add to the index (see next slide).

```
$ git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be
committed)

    new-file

nothing added to commit but untracked files present (use "git
add" to track)
$ git add new-file
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   new-file
$
```

git add --patch/-p

Update the index with specific changes from the working tree.

Splits changes between the index and working tree into individual *hunks* (groups of changed lines), and asks you whether each one should be added to the index.

"y": Includes the hunk
"n": Omits the hunk
"s": Splits the hunk into smaller ones
"?": Explains other options

```
$ git add -p
diff --git a/directory2/entryway b/directory2/entryway
index d5ba6ac..4765745 100644
--- a/directory2/entryway
+++ b/directory2/entryway
<...>
@@ -33,4 +29,5 @@ shoot you."
 differences, but I had hoped to clear them up tonight by
inviting him. I did
 not kill him! Please help clear my name!"

-You make your way into the living room.
+"Perhaps that would be easier to do were you not named David
Knifehands," you
+suggest, and brush past David into the living room.
Stage this hunk [y,n,q,a,d,K,g,/,e,?]? y

$
```

`git rm`

Remove a file from the index *and* working tree.

Note that this deletes the file on disk as well as in the index, unlike `git add`, which only ever changes the index. Won't delete a file with uncommitted changes without `-f`.

`--cached`: Removes the file from the index, but leaves the working tree copy.

`-r`: Remove a directory and all files in it.

```
$ git rm file1
rm 'file1'
$ ls
directory1  directory2  file2  file3  missing-link

$ git rm directory2/entryway error: the following file has
changes staged in the index: directory2/entryway (use
--cached to keep the file, or -f to force removal)

$ git status -s
git status -s
 M directory2/entryway
D  file1
$
```

git mv

Rename a file in the working tree and index.

Git cannot represent renames in commits, but this is a convenient way to remove a file from the index and add the same file with a different name.

`--cached`: Alters the index but not the working tree.

`--force (-f)`: Moves the file in the working tree even if it will overwrite an existing file.

```
$ git mv file3 directory1/
$ git status
On branch main
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    file3 -> directory1/file3

$
```

`git restore`

Restore individual files to versions from a previous commit.

Note that this command does *not* change HEAD to the given commit. That means that the restored contents will appear as unstaged changes.

Note: This command is new and may not exist for you.

`--source` (`-s`): The commit to restore to.

`--patch` (`-p`): Interactively select hunks to restore.

```
$ git log --oneline file2
8320c08 Fix file2 to match file zoo
fe7b7f5 Add file2

$ git restore -s fe7b7f5 file2
$ git diff
diff --git a/file2 b/file2
index a5c1966..48d5349 100644
--- a/file2
+++ b/file2
@@ -1 +1 @@
-Hello, world
+hello wodrl
$
```

`git checkout <commit> <file...>`

Like `git restore`, but more widely available.

Note that the list of files after the commit are crucial. If you omit them, the command does something else (detailed in a later slide). This is why `git restore` was introduced as a separate command.

Unlike `git restore`, adds restored changes to the index.

`--patch` (`-p`): Interactively select hunks to restore.

```
$ git checkout fe7b7f5 file2
Updated 1 path from d5f1e5a
$ git diff --cached
diff --git a/file2 b/file2
index a5c1966..48d5349 100644
--- a/file2
+++ b/file2
@@ -1 +1 @@
-Hello, world
+hello wodrl
$
```