How Do I…?

This final chapter presents some commands and recipes for accomplishing a grab bag of specific tasks. Some were presented earlier and are repeated or referred to here for easy reference, and some are new. Remember that you don't usually want to edit history for commits you've already published with `git push`. Examples that refer to a remote repository use the most common case, `origin`. `rev` is any revision name as described in Chapter 8.

…Make and Use a Central Repository?

Suppose you have an account named `ares` on a server `mars.example.com`, which you want to use to coordinate your own work on a project `foo` (perhaps among repositories at home, work, and on your laptop). First, log into the server and create a "bare" repository (which you will not use directly):

```
$ ssh ares@mars.example.com
ares> git init --bare foo
Initialized empty Git repository in /u/ares/foo/.git
$ logout
```

If this is for a project with existing content, connect that repository to the new remote as its origin (assuming here a single, local *master* branch):

```
$ cd foo
$ git remote add origin ares@mars.example.com:foo
```

```
$ git push -u origin master
...
To ares@mars.example.com:foo
* [new branch]        master -> master
Branch master set up to track remote branch master
from foo.
```

You can just use plain `git push` from then on. To clone this repository elsewhere:

```
$ git clone ares@mars.example.com:foo
```

. . . Fix the Last Commit I Made?

Make your corrections and stage them with `git add`, then:

```
$ git commit --amend
```

Add `-a` to automatically stage all changes to tracked files (skipping `git add`). Add `-C HEAD` to reuse the previous commit message without stopping to edit it.

See "Changing the Last Commit" on page 58.

. . . Edit the Previous n Commits?

```
$ git rebase -i HEAD~n
```

The history involved should be linear. You can add `-p` to preserve merge commits, but this can get tricky depending on the changes you want to make.

See "Editing a Series of Commits" on page 64.

. . . Undo My Last n Commits?

```
$ git reset HEAD~n
```

This removes the last *n* commits of a linear history from the current branch, leaving the corresponding changes in your working files. You can add `--hard` to make the working tree reflect the new branch tip, but beware: this will also discard any current uncommitted changes, which you will lose with no recourse. See

"Discarding Any Number of Commits" on page 61. This will also work if there is a merge commit in the range, effectively undoing the merge for this branch; see Chapter 8 to understand how to interpret `HEAD~n` in this case.

. . . Reuse the Message from an Existing Commit?

```
$ git commit --reset-author -C rev
```

Add `--edit` to edit the message before committing.

. . . Reapply an Existing Commit from Another Branch?

```
$ git cherry-pick rev
```

If the commit is in a different local repository, `~/other`:

```
$ git --git-dir ~/other/.git format-patch ↵
-1 --stdout rev | git am
```

See:

- "Importing Linear History" on page 156
- "git cherry-pick" on page 181

. . . List Files with Conflicts when Merging?

`git status` shows these as part of its report, but to just list their names:

```
$ git diff --name-only --diff-filter=U
```

. . . Get a Summary of My Branches?

- List local branches: `git branch`

- List all branches: `git branch -a`
- Get a compact summary of local branches and status with respect to their upstream counterparts: `git branch -vv`
- Get detail about the remote as well: `git remote show origin` (or other named remote)

See "Notes" on page 91.

. . . Get a Summary of My Working Tree and Index State?

```
$ git status
```

Add `-sb` for a more compact listing; see the "Short Format" section of *git-status(1)* on how to interpret this.

. . . Stage All the Current Changes to My Working Files?

```
$ git add -A
```

This does `git add` for every changed, new, and deleted file in your working tree. Add `--force` to include normally ignored files; you might do this when adding a new release to a "vendor branch," which tracks updates to other projects you obtain by means other than Git (e.g., tarballs).

. . . Show the Changes to My Working Files?

`git diff` shows unstaged changes; add `--stage` to see staged changes instead. Add `--name-only` or `--name-status` for a more compact listing.

. . . Save and Restore My Working Tree and Index Changes?

`git stash` saves and sets your outstanding changes aside, so you can perform other operations that might be blocked by them, such as checking out a different branch. You can restore your changes later with `git stash pop`. See "git stash" on page 188.

. . . Add a Downstream Branch Without Checking It Out?

```
$ git branch foo origin/foo
```

This adds a local branch and sets up push/pull tracking as if you had done `git checkout foo`, but does not do the checkout or change your current branch.

. . . List the Files in a Specific Commit?

```
$ git ls-tree -r --name-only rev
```

This listing is restricted to the current directory; add `--full-tree` for a complete list.

. . . Show the Changes Made by a Commit?

`git show` *rev* is easier that `git diff` *rev*`~` *rev*, and shows the author, timestamp, commit ID, and message as well. Add `-s` to suppress the diff and just see the latter information; use `--name-status` or `--stat` to summarize the changes. It also works for merge commits, showing conflicts from the merge as with `git log --cc` (see "Showing Diffs" on page 142). The default for `rev` is HEAD.

. . . Get Tab Completion of Branch Names, Tags, and So On?

Git comes with a completion package for *bash* and *zsh*, installed in its `git-core` directory as `git-completion.bash`. You can use it by including (or "sourcing") this file in your shell startup file (e.g., in `~/.bashrc`):

```
# define completion for Git
gitcomp=/usr/share/git-core/git-completion.bash
[ -r $gitcomp ] && source $gitcomp
```

Pressing Tab in the middle of a Git command will then show possible completions for the given context. For example, if you type `git checkout`, space, and then press Tab, the shell will print the branches and tag names you could use here. If you type an initial part of one of these names, pressing Tab again will complete it for you. The exact behavior of completion is very customizable; see your shell manpage for details.

There is also a `git-prompt.sh`, which will make your shell prompt reflect the current branch status when your working directory is a Git repository.

. . . List All Remotes?

`git remote` does this; add `-v` to see the corresponding URLs configured for push and pull (ordinarily the same):

```
$ git remote -v
origin  http://olympus.example.com/aphrodite (fetch)
origin  http://olympus.example.com/aphrodite (push)
```

. . . Change the URL for a Remote?

```
$ git remote set-url remote URL
```

. . . Remove Old Remote-Tracking Branches?

```
$ git remote prune origin
```

This removes tracking for remote branches that have been deleted upstream.

. . . Have git log:

Find Commits I Made but Lost?

. . . perhaps after editing history with `git rebase -i` or `git reset`, or deleting a branch:

```
$ git log -g
```

See "Double Oops!" on page 59.

Not Show the diffs for Root Commits?

A root commit always shows the addition of all the files in its tree, which can be a large and uninformative list; you can suppress this with:

```
$ git config [--global] log.showroot false
```

Show the Changes for Each Commit?

`git log -p` shows the complete patch for each commit it lists, while these options summarize the changes in different ways:

```
$ git log --name-status
$ git log --stat
```

See "Listing Changed Files" on page 136.

Show the Committer as well as the Author?

```
$ git log --format=fuller
```