Git In Here

An Unoffical Introduction To Git Version Control Management

Sam Dixon
https://github.com/neaop/git_in_here
s.dixon@napier.ac.uk

2018

Contents

What is Version Control? I

Version Control Systems are tools that let you track changes
A number of different VCSs are available:

- Subversion
- Mercurial
- CVS
- Fossil
- Git

Each VCS has its own quirks in style and implementation

What is Version Control? II

Common features of VCSs include:

- Timeline of changes
- File reversion
- Branching
- Tagging
- Collaboration

Why you should use Version control

VCSs can help with common problems like:

- Creating multiple copies files “in case you need the old one”
- Put work onto a USB drive to move it between computers
- Copied code into email/IM to send it to a friend
- Lost work due to loss of power/faulty hard drive
- Overwritten someone else’s work when collaborating

What is Git? I

Git was developed in 2005 by Linus Torvalds.
It has become one of the most popular VCSs in use today
Some companies that use Git include:

- Google
- Apple
- Microsoft
- Facebook
- Amazon
- Adobe
- Mozilla
- NASA

What is Git? II

Git has a number of key features that make it an attractive VCS

- Distributed
- Decentralised
- Free (like beer *and* like speech)
- Multiplatform
- Speed
- Popularity

Common Terms

Git has a number of key terms that you should keep in mind

- Repository
- Commit
- Push/Pull
- Branch
- Merge

Some of these terms are shared with other VCSs

Setting up Git I

Git can be installed in a number of ways:

- Download directly: `https://git-scm.com/downloads`
- Install a Git visualiser (GitHub Desktop, Sourcetree, Kraken)
- Install via a package manager

Setting up Git II

Git requires a name and email to create commits

```
$ git congif --global user.name "Sam"
$ git config --global user.email sam@mail.com
```

These settings are not used as login credentials / tokens

Git Workflow

The standard process for using Git is as follows:

1. **Initialise** a Git repo
2. Do some work (edit files)
3. **Stage** the changes you want to keep
4. Finalise the changes in a **Commit**
5. Repeat from step 2

Getting a Repo

Git can create a repo in a new folder:

```
$ git init project
```

Or in an existing directory:

```
$ git init
```

It doesn't matter if there are files already in the directory

Staging Files

Git will not track files by default - you have to specify:

```
$ git add file_name
```

You can add multiple files in a single command:

```
$ git add file_1 file_2
```

Or you can use glob wildcards:

```
$ git add *.txt
```

Checking File State

You can check the state of a repo with the following:

```
$ git status
```

Files can be in multiple states:

- Untracked
- Modified
- Staged
- Deleted

You can see how a tracked file has been modified:

```
$ git diff file_name
```

Committing Changes

When you are ready to finalise your changes; commit:

```
$ git commit
```

You can also declare a commit message in the same command:

```
$ git commit -m "Your message here"
```

Viewing History

Once you have a few commits, you can review them:

```
$ git log
```

View what changes were made in each commit with the `patch` flag:

```
$ git log -p
```

There are multiple flags that can alter how the log looks:

```
$ git log --oneline --graph --decorate
```

Ignoring Files

You may not want to track all the files in repo
Config files, binary files and build files are often unwanted

You can instruct git to ignore file using a git-ignore file:
Create a file called .gitignore in the root of your repo
Add glob patterns for files you want to ignore

`/bin` - ignores all files in the bin dir
`*.o` - ignores all object files

Many sample git-ignore files are available online

Undoing

The `amend` flag allows you to alter a commit
It adds your staged files to the *previous* commit
Use this to add files you forgot, or for quick alterations:

```
$ git commit
$ git add forgotten_file
$ git commit --amend
```

When you amend a commit you can also change it's message

Unstaging a File

You may accidentally stage a file that you do not want to commit
You can unstage the file without losing the changes you made:

```
$ git reset HEAD file_name
```

`HEAD` is Git's name for the tip of the current branch

Unmodifying a File

You may edit a file and decide to not keep the modifications
You revert a file to how it was when it was last committed:

```
$ git checkout -- file_name
```

This command is dangerous - you cant recover uncommitted changes

Reverting a Commit

A main feature of a VCS is the ability to undo previous changes

Git provides this function with the `revert` command

First use the `log` command to display commit hashes

You can then undo the changes that a commit introduced:

```
$ git revert commit_hash
```

Git will then create a commit inverse of the selected hash

Reverting a Series of Commits

You can also undo a series of commits in a single command

Use `log` to find the first and last commit to remove

```
$ git revert from_hash..to_hash
```

Use the `--no-edit` flag for git if you don't want to change the default revert message

Branching

Branches are a key feature of Git

They allow different versions of a project to exists simultaneously

Allows for parallel development of features

Branches can be used to experiment or write patches

Branches are inexpensive to create/delete

All repos start with a branch named `master`

Creating Branches

You can quickly create branches:

```
$ git branch branch_name
```

This will create a new branch starting at your current commit

You can list the different branches in a repo:

```
$ git branch
```

The `status` command will inform you of which branch you are on

Swapping branches

Once a branch has been created, you need to switch to it:

```
$ git checkout branch_name
```

Git may stop you from swapping if you have uncommitted changes

Merging Branches

You can add the changes from one branch to another
Merging will make Git apply the divergent commits from a branch

First, swap to the branch you want to apply the commits to:

```
$ git checkout master
```

Then merge in your branch with the commits you want:

```
$ git merge branch_name
```

Conflicts

Sometimes you will not be able to merge due to conflicts

Conflicts occur when two branches edit a file differently

Conflicts must be resolved before a merge completes

The `status` command informs you which files have problems

Resolving Conflicts

A simple way to fix a conflict is to simply open the file

Git will automatically tag the conflicting area

You can edit the file to keep the version you want

When you a happy with the file, stage it and `commit`

Remotes I

Git can be used entirely locally

But remotes allow you to collaborate and store projects

A remote is copy of a Git repo “somewhere else”

A number of services will host repos for you

You can even run your own Git server

Remotes II

There a two ways to set up repo remotes

You can make a local copy of a remote repo:

```
$ git clone remote_url
```

This will copy the repo and set the url as the default remote

You can also add a remote to a local repo manually:

```
$ git remote add remote_name remote_url
```

The default name for a repo's primary remote is `origin`

Remotes III

You can see what remotes are configured to a repo:

```
$ git remote
```

To see the url's of theses remotes include the `verbose` flag:

```
$ git remote -v
```

You can rename remote:

```
$ git remote rename remote_name new_name
```

Or remove it if you dont need to use it any more:

```
$ git remote remove remote_name
```

Remotes IV

You can share local commits to a remote:

```
$ git push remote_name branch_name
```

The default branch name is `master`

You can query a remote to learn its state:

```
$ git remote show remote_name
```

Remotes V

To get commits from a remote repo, you must fetch it:

```
$ git fetch remote_name
```

Fetching will copy data to a branch - local files are unchanged

You can inspect these branches with `checkout` and `log`

Remotes VI

You can add changes from a remote branch with merge:

```
$ git merge remote_branch_name
```

You can revert theses merges if you don't like them

Useful Links

Pro Git Book: `https://git-scm.com/book/en/v2`

Atlassian Git Tutorials:
`https://www.atlassian.com/git/tutorials`

GitHub interactive Tutorial:
`https://try.github.io/levels/1/challenges/1`

Sample git-ignore files:
`https://github.com/github/gitignore`