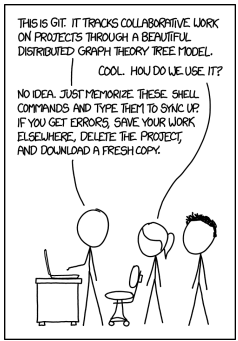
Git

Jonathan Hodgson (Archie)



August 7, 2020

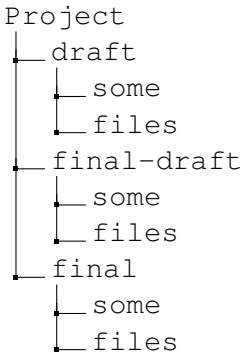
What is Git

A very versatile Version Control System

- Keep track of source code (or other folders and files) and its history
- Facilitate collaboration
- Distributed

Git $\neq$ Github

Naïve Approach



Pros

- Simple
- No dependencies
- No Learning curve

Cons

- Difficult to collaborate
- Lot's of wasted disk space
- Hard to find particular versions of files

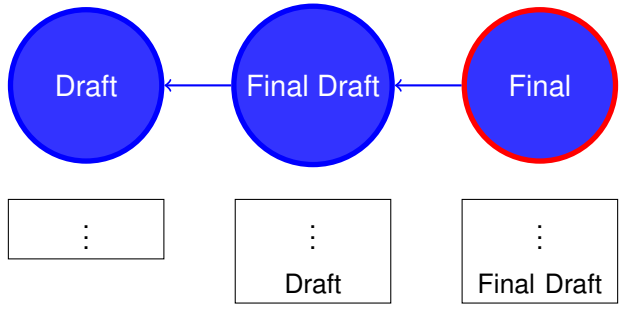
Files and Folders

Blob In Git, a file is called a blob.

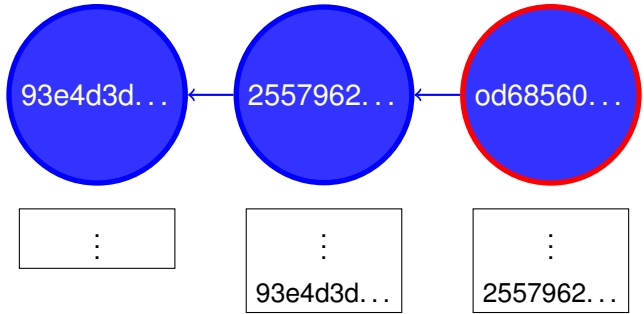
Tree In Git, a directory is called a tree.

Commits

(Snapshots)



Commits



Staging Area

- Sometimes called the git index
- An intermediate area in which you can pick files to be included in the next commit.
- Also allows you to exclude some files from your version history.
  - Log files
  - Binary files
  - Minified files

Install

```
# Ubuntu / Debian / Kali
sudo apt install git

# Centos / Fedora / Red Hat
sudo dnf install git

# Arch / Antergos / Manjaro
sudo pacman -S git

# Mac
brew install git

# Get the Version
git --version
```

Git for Windows: https://gitforwindows.org/

Setting It Up

User

```
git config --global user.name "Jonathan Hodgson"
git config --global user.email "git@jonathanh.co.uk"
```

Setting It Up

Editor

Pick One

```
# Set editor to vim
git config --global core.editor "vim"

# Set editor to nano
git config --global core.editor "nano"

# Set editor to VS Code
git config --global core.editor "code -w"

# Set editor to Sublime
git config --global core.editor "subl -w"
```

Create a repository

```
❯ mkdir /tmp/demo
❯ cd /tmp/demo
❯ git init
Initialized empty Git repository in /tmp/demo/.git/
❯ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Git status

```
touch greeting.py
chmod +x !$
vim greeting.py
git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    greeting.py

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

Staging Area

```
# Add files / or directories
git add <file|directory> [<file|directory>...]
# Add everything not in gitignore
git add -A
```

Staging Area

```
❯ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    greeting.py

nothing added to commit but untracked files present (use "git add" to track)
❯ git add greeting.py
❯ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file: greeting.py
```

Committing

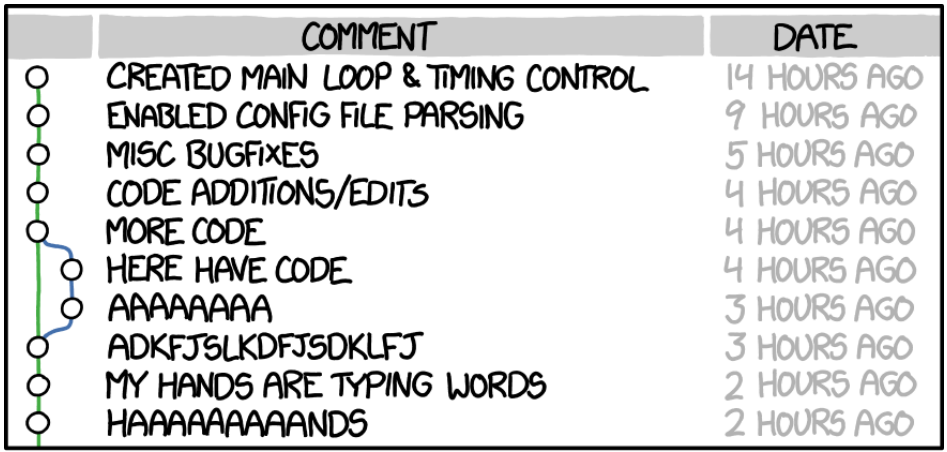
- First line should be concise summary around 50 chars
- Body Should be wrapped to around 70 chars
- There should be an empty line separating summary from body
- If contributing to a project, check per-project guidelines
  - Normally in contributing.md or similar
- Use the imperative: “Fix bug” and not “Fixed bug” or “Fixes bug.”

When should you commit?

Commit early, commit often

- Every time you complete a small change or fix a bug
- For each point on a detailed to-do list
- You don't normally want to commit broken code (intentionally at least)
- In some instances you might want to auto-commit - but probably not too often.
  - Normally this works if changes can't break something. E.g. Password Manager

Commit Messages



AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Commit

```
# Open editor for message
git commit
# Read message from file
git commit -F <file or - for stdin>
# Provide message directly
git commit -m "<message>"
```

```
git commit
[master (root-commit) 9d31a8d] Add greeting.py
 1 file changed, 7 insertions(+)
 create mode 100755 greeting.py
```

Diff

```
# Diff between last commit and current state
git diff
# Diff between 2 commits or references
git diff commit1..commit2
# Same as above but on a single file
git diff a/file
```

Diff

```
 git diff
diff --git a/greeting.py b/greeting.py
index 451f386..e73cd5b 100755
--- a/greeting.py
+++ b/greeting.py
@@ -1,7 +1,7 @@
 #!/usr/bin/env python

 def main():
- print("Hello")
+     print("Hello World")

 if __name__ == "__main__":
     main()
```

Log

```
❯ git commit -m "Change \"Hello\" to \"Hello World\""
[master be729cc] Change "Hello" to "Hello World"
 1 file changed, 1 insertion(+), 1 deletion(-)
...
❯ git log
commit be729ccc29cea76ea6419527e9d79641d7882182
Author: Jonathan Hodgson <git@jonathanh.co.uk>
Date: Fri Aug 7 13:43:21 2020 +0100

    Change "Hello" to "Hello World"

commit 9d31a8dd7817c1d12592219217115c5b33437ffa
Author: Jonathan Hodgson <git@jonathanh.co.uk>
Date: Fri Aug 7 13:43:19 2020 +0100

    Add greeting.py

    Adds the first file, currently always prints Hello
```

Under the hood

```
❯ zlib-flate -uncompress < .git/objects/be/729ccc29cea76ea6419527e9d79641d7882182
commit 256tree a3a29fe10acf63f53164292740d22d530750d9ab
parent 9d31a8dd7817c1d12592219217115c5b33437ffa
author Jonathan Hodgson <git@jonathanh.co.uk> 1596804201 +0100
committer Jonathan Hodgson <git@jonathanh.co.uk> 1596804201 +0100

Change "Hello" to "Hello World"
❯ zlib-flate -uncompress < .git/objects/be/729ccc29cea76ea6419527e9d79641d7882182 | sha1sum
be729ccc29cea76ea6419527e9d79641d7882182
❯ git cat-file -p a3a29fe
100755 blob e73cd5b9fd440608e22b70411a55645e3611fa15 greeting.py
❯ git cat-file -p e73cd5b
#!/usr/bin/env python

def main():
    print("Hello World")

if __name__ == "__main__":
    main()
```

.gitignore

This file tells git which files not to track.

```
*.log
*.doc
*.pem
*.docx
*.jpg
*.jpeg
*.pdf
*.png
.DS_Store/
*.min.css
*.min.js
dist/
```

References

- We have just seen that commits are simply (compressed) text files, addressed by a hash.
- References are a way of addressing them without remembering the hash.
- Unlike the hashes, references can change - and they do change.

References

- Branches
  - Parallel development
- Tags
  - Special points in history (Release versions)
- HEAD
  - Current position in history

References

```
git log
commit be729ccc29cea76ea6419527e9d79641d7882182 (HEAD -> master)
Author: Jonathan Hodgson <git@jonathanh.co.uk>
Date: Fri Aug 7 13:43:21 2020 +0100

    Change "Hello" to "Hello World"

commit 9d31a8dd7817c1d125922192171l5c5b33437ffa
Author: Jonathan Hodgson <git@jonathanh.co.uk>
Date: Fri Aug 7 13:43:19 2020 +0100

    Add greeting.py

    Adds the first file, currently always prints Hello
```

References

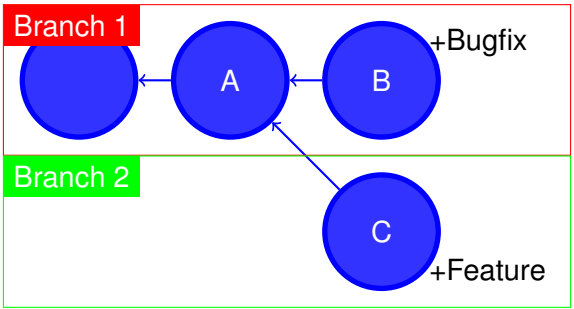
```
cat .git/refs/heads/master
be729ccc29cea76ea6419527e9d79641d7882182
cat .git/HEAD
ref: refs/heads/master
```

- References are stored in the `.git/refs` folder
- The `heads` folder contains references to the heads (or tips) of all local branches

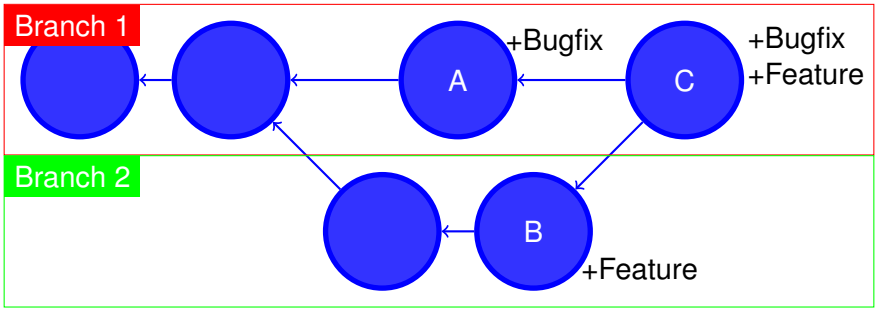
Branches

- By default Git will create a branch called Master (maybe?).
- Allows multiple features to be developed in parallel without interference.
- Allows multiple people to collaborate easily.

Commits / Branches



Commits / Branches



```
# List Branches
git branch # -v adds more info

# Create a branch called test
git branch test # or
cp ~/.git/refs/heads/master ~/.git/refs/heads/test

# Switch to new branch
git switch test # or
git checkout test

# Create and switch in one go
git switch -c test # or
git checkout -b test
```

Branches

```
❯ git branch -v
* master be729cc Change "Hello" to "Hello World"
❯ git switch -c test
❯ git branch -v
  master be729cc Change "Hello" to "Hello World"
* test   be729cc Change "Hello" to "Hello World"
❯ git log --oneline --all
be729cc (HEAD -> test, master) Change "Hello" to "Hello World"
9d31a8d Add greeting.py
```

Differing Branches

```
❯ git switch master
❯ vim greeting.py
  # CAPITALISE HELLO WORLD #
❯ git commit -am "Capitalises Hello World"
[master d7e14c5] Capitalises Hello World
 1 file changed, 1 insertion(+), 1 deletion(-)
❯ git switch test
❯ vim greeting.py
  # Adds the line "import sys" #
❯ git commit -am "Adds sys import for arg parsing"
[test fdb566e] Adds sys import for arg parsing
 1 file changed, 2 insertions(+)
```

Differing Branches

```
git log --oneline --all --graph
* d7e14c5 (master) Capitalises Hello World
| * fdb566e (HEAD -> test) Adds sys import for arg parsing
|/
* be729cc Change "Hello" to "Hello World"
* 9d31a8d Add greeting.py
git diff mater..test
diff --git a/greeting.py b/greeting.py
index 483ed66..a0ab589 100755
--- a/greeting.py
+++ b/greeting.py
@@ -1,7 +1,9 @@
 #!/usr/bin/env python

+import sys
+
 def main():
-    print("HELLO WORLD")
+    print("Hello World")

 if __name__ == "__main__":
     main()
```

Simple Merge

```
❯ git switch master
❯ git merge test
Auto-merging greeting.py
Merge made by the 'recursive' strategy.
 greeting.py | 2 ++
 1 file changed, 2 insertions(+)
❯ git log --oneline --all --graph
*   3fec4b4 (HEAD -> master) Merge branch 'test' into master
|\
| * fdb566e (test) Adds sys import for arg parsing
* | d7e14c5 Capitalises Hello World
|/
* be729cc Change "Hello" to "Hello World"
* 9d31a8d Add greeting.py
```

Tidy Up

```
git switch master
git branch -d test
Deleted branch test (was fdb566e).
```

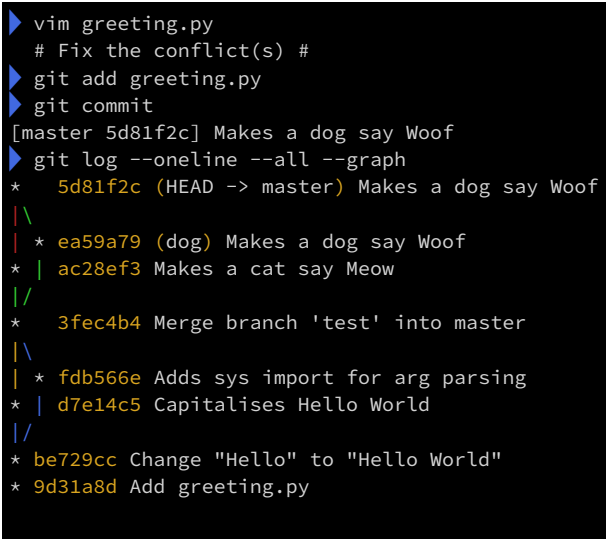
More Complex merge

```
  # Make changes to 2 branches in the same place #
❯ git switch master
❯ git log --oneline --all --graph
* ea59a79 (dog) Makes a dog say Woof
| * ac28ef3 (HEAD -> master) Makes a cat say Meow
|/
*   3fec4b4 Merge branch 'test' into master
|\
| * fdb566e Adds sys import for arg parsing
* | d7e14c5 Capitalises Hello World
|/
* be729cc Change "Hello" to "Hello World"
* 9d31a8d Add greeting.py
❯ git merge dog
Auto-merging greeting.py
CONFLICT (content): Merge conflict in greeting.py
Automatic merge failed; fix conflicts and then commit the result.
```

More Complex merge

```
cat greeting.py
#!/usr/bin/env python

import sys

<<<<<<< HEAD
def cat():
    print("Meow")

def main():
    if len(sys.argv) > 1 and sys.argv[1] == "cat":
        cat()
=======
def dog():
    print("Woof")

def main():
    if len(sys.argv) > 1 and sys.argv[1] == "dog":
        dog()
>>>>>>> dog
    else:
        print("HELLO WORLD")

if __name__ == "__main__":
    main()
```

More Complex merge



Time Travel

```
# Print a version of a file
git show <commit or reference>:<file>
# Restore a file from a previous version
git restore -s <commit or reference> file # or
git checkout <commit or reference> -- file
# Go back in time to a commit
git switch --detach <commit or reference> # or
git checkout <commit or reference>
```

Remotes

- The majority of Git commands only affect your local repository.
- Git has a concept called remotes which you can think of as other instances of the same repository
- Git has a selection of commands that are used to communicate with these remote repositories
- It can communicate on multiple protocols including
  - HTTP(S)
  - SSH
  - Local Filesystem

Adding a remote

```
❯ git remote add origin /tmp/demo-remote
❯ git remote
origin
❯ git remote get-url origin
/tmp/demo-remote
```

Pushing your code

Long Way

```
git push <remote> <local-branch>:<remote-branch>
# E.g.
git push origin master:master
```

Pushing your code

Easy way

```
❯ git branch --set-upstream-to=origin/master master
❯ git push
❯ git status
On branch master
nothing to commit, working tree clean
```

Retrieving changes from the remote

```
❯ git fetch
❯ git status
On branch master
nothing to commit, working tree clean
❯ git log --oneline --all --graph
* 7515e94 (origin/master) Adds Cow option
*   5d81f2c (HEAD -> master) Makes a dog say Woof
|\
| * ea59a79 (dog) Makes a dog say Woof
* | ac28ef3 Makes a cat say Meow
|/
*   3fec4b4 Merge branch 'test' into master
|\
| * fdb566e Adds sys import for arg parsing
* | d7e14c5 Capitalises Hello World
|/
* be729cc Change "Hello" to "Hello World"
* 9d31a8d Add greeting.py
❯ git merge
```

Git Pull

Shortcut

```
git pull
git pull <remote> <branch>
```

Cloning

```
# Clone a repository into a folder
git clone <URL> <folder>

# Clone a repository into a folder on a specific branch
git clone --branch <branch> <URL> <folder>

# Shallow clone a repository into a folder
git clone --shallow <URL> <folder>
```

Issues

- Not part of Git, rather something most Git hosting providers offer.
- You can normally reference issues in commit messages using # symbol.

Pull Requests

Merge Requests

1. Fork
2. Clone
3. Branch
4. Commit
5. Push

Useful supporting tools

Shell Integration

Git ships with completion for bash, zsh and tcsh. You may need to source it in the relevant rc file.

Prompt customisation is available out of the box for bash and zsh.

Useful supporting tools

Editor Plugin

- Git Gutters
- Easy staging of parts of a file
- Merge Conflict Resolution

Useful supporting tools

BFG Repo Cleaner

You'll need something like this when you realise you have just committed your ssh keys

https://rtyley.github.io/bfg-repo-cleaner/

Useful supporting tools

Git Crypt

For storing sensitive information in a git repository.

https://github.com/AGWA/git-crypt

Useful supporting tools

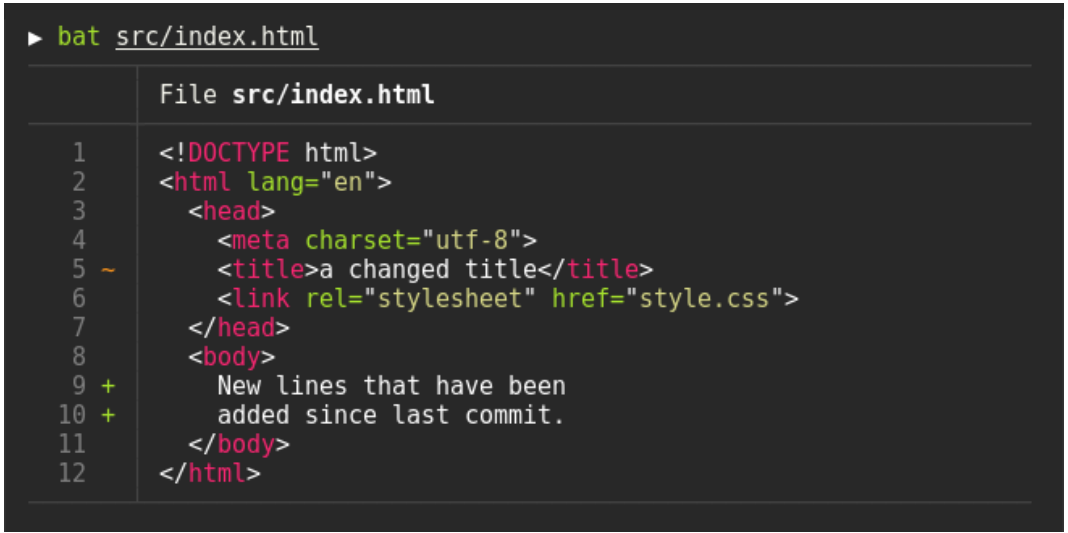
Git Large File Storage

A solution to the issue of storing binary files

https://git-lfs.github.com/

Useful supporting tools

Bat



Useful supporting tools

RipGrep / Fd / Exa

FD replaces find

https://github.com/sharkdp/fd

RigGrep replaces grep

https://github.com/BurntSushi/ripgrep

Exa replaces ls

https://github.com/ogham/exa

Useful supporting tools

Pass

- Password Manager
- Uses Git for keeping track of history
- Syncs using Git
- Everything is encrypted with a GPG key
- Has compatible android, ios and browser apps.

https://www.passwordstore.org/

Useful supporting tools

tldr

The man page for git pull is over 700 lines.