



Everything you want to know about

Git

A TECH TALK BY BEN CHAPMAN-KISH

MARCH 29, 2018

Topics

- ▶ Intro
- ▶ Common Git workflows
- ▶ Commits and branches
- ▶ More commands and tools
- ▶ Tips and tricks
- ▶ How Git *really* works

What is Git, again?

- ▶ Distributed version control system
- ▶ Created by Linus Torvalds in 2005
- ▶ Open-source, well-maintained
- ▶ High performance, powerful, secure



Everyday use of Git

- ▶ Pull
- ▶ Push
- ▶ Checkout
- ▶ Branch
- ▶ Merge
- ▶ Add
- ▶ Commit
- ▶ Tag
- ▶ Status
- ▶ Diff

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
~/Documents/git-example Pmaster 1 git pull ✓
Updating f0ae20f..9043d66
Fast-forward
 server.py | 35 ++++++
 start.py  | 16 ++++++
 2 files changed, 47 insertions(+), 4 deletions(-)
 create mode 100644 server.py
~/Documents/git-example Pmaster vim server.py ✓
~/Documents/git-example Pmaster git status ✓
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   server.py

no changes added to commit (use "git add" and/or "git commit -a")
~/Documents/git-example Pmaster git add . ✓
~/Documents/git-example Pmaster git commit -m "Fix bug in server" ✓
[master 22817b5] Fix bug in server
1 file changed, 11 insertions(+), 8 deletions(-)
~/Documents/git-example Pmaster 1 git push ✓
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 335 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/BenChapmanKish/git-examples.git
 9043d66..22817b5 master -> master
~/Documents/git-example Pmaster
```

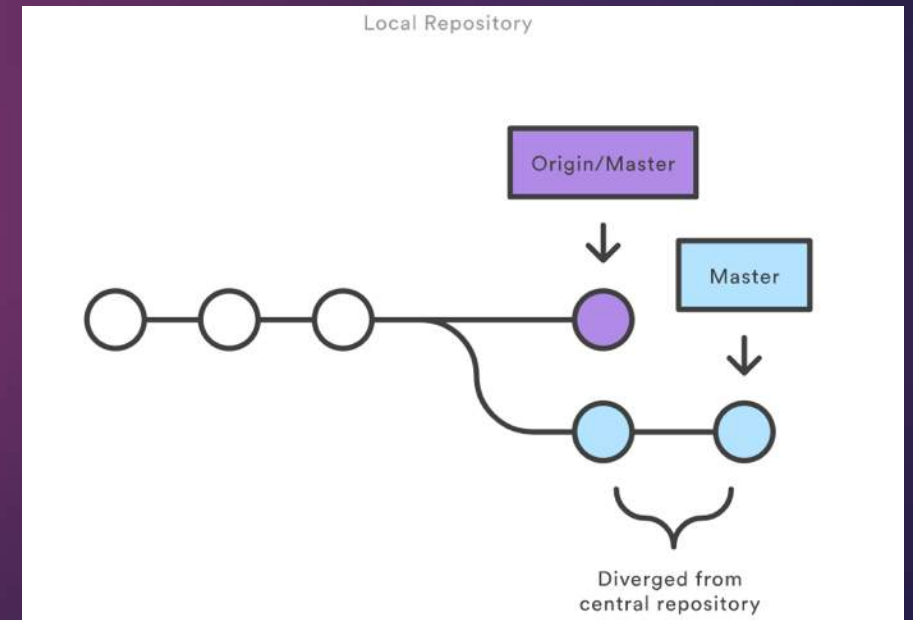
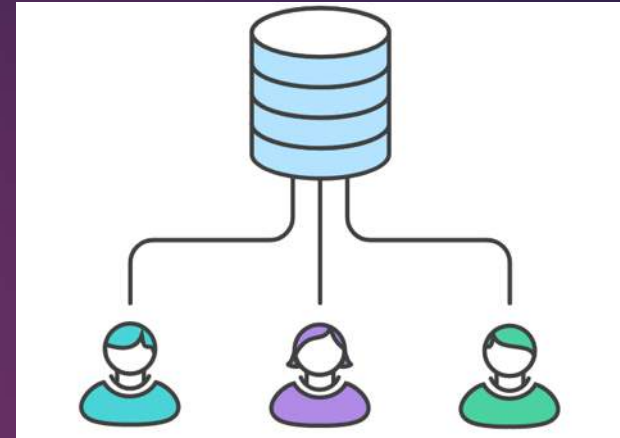
... among many others, depending on your workflow



Common Git Workflows

Basic/Centralized

- ▶ Essentially only one branch on one central repository
- ▶ Every dev makes their own local changes and pulls/pushes as necessary
- ▶ Ideal for small teams or teams transitioning from SVN



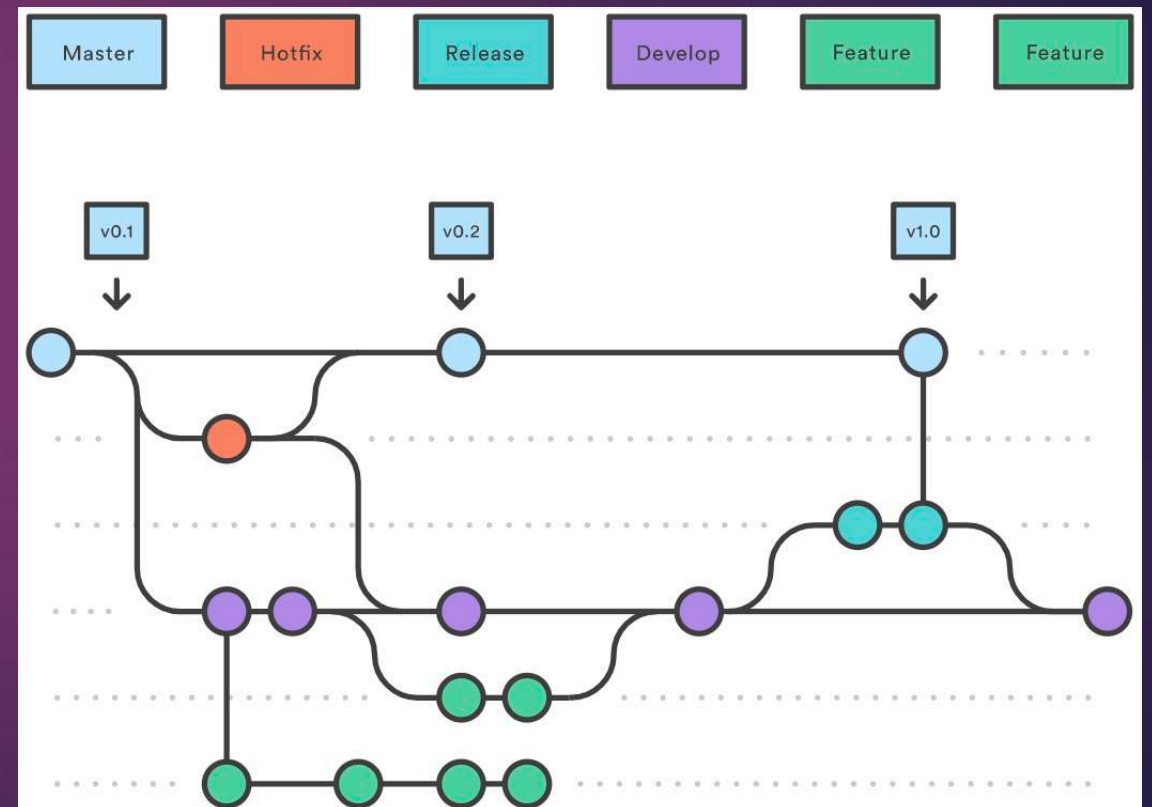
Feature branches

- ▶ Single branch with infinite lifetime – master
- ▶ Branch off for each feature, then merge/rebase back in when it's done
- ▶ Easier to work on concurrent features than centralized workflow



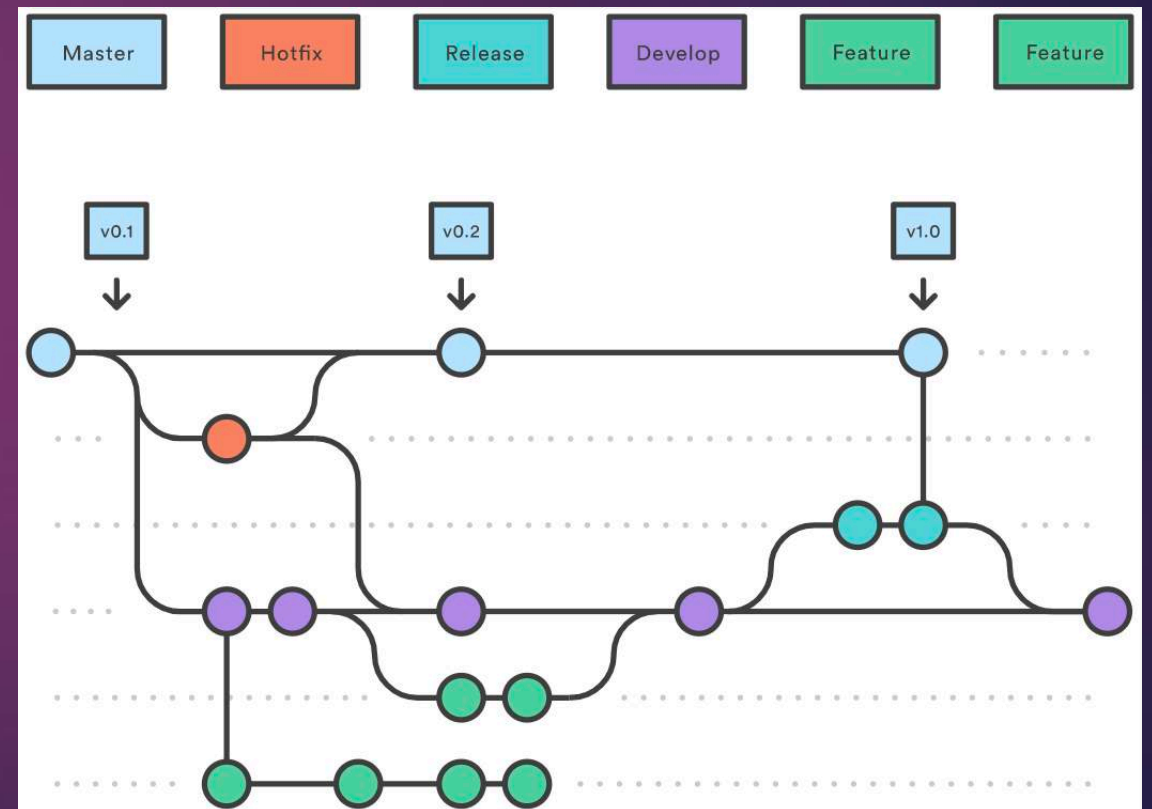
Git Flow

- ▶ Two parallel branches with infinite lifetime – master and develop
- ▶ Master is production-ready code
- ▶ Develop is integration branch for all features
- ▶ Feature branches are branched off and merge back into develop



Git Flow

- ▶ Release branches branch off develop, only bug fixes are made
- ▶ When it's ready, the release branch merges into master and develop
- ▶ Hotfix branches branch off and back into master to quickly fix bugs in prod



Git Flow

- ▶ Ideal for large or release-based projects
- ▶ Makes parallel development very easy
- ▶ Offers release staging area for fixes and testing before shipping
- ▶ Offers dedicated channel for hotfixes to production

Forking

- ▶ Each dev has their own “fork” of the official repo
- ▶ Changes are made on a new branch in the forked repo
- ▶ When the changes are done, a pull request is made from the new branch to the official repo
- ▶ Often used in public open-source projects and in conjunction with hosting services such as GitHub



Before we go any further...

Commits and Branches

What exactly is a commit?

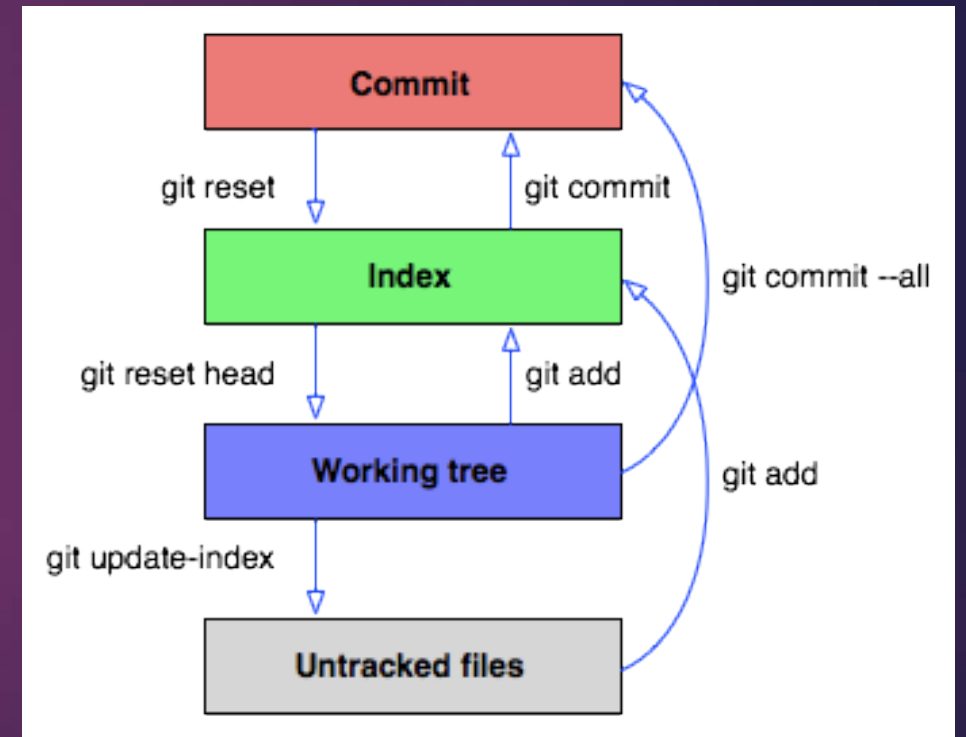
- ▶ A commit is a snapshot of the state of the project at a certain point
- ▶ Commits also contain metadata, such as:
 - ▶ The commit message
 - ▶ The author of the commit
 - ▶ The time the commit was made
 - ▶ Its parent commit(s), if any
- ▶ Every commit can be uniquely identified by its SHA-1 hash

What exactly is a commit?

- ▶ A commit's hash is generated from the “snapshot” and its metadata
- ▶ When you change a commit, you're really making a new commit with new data and a new hash
- ▶ If the changed commit has children, every child commit will have to be recreated with their parent hashes updated

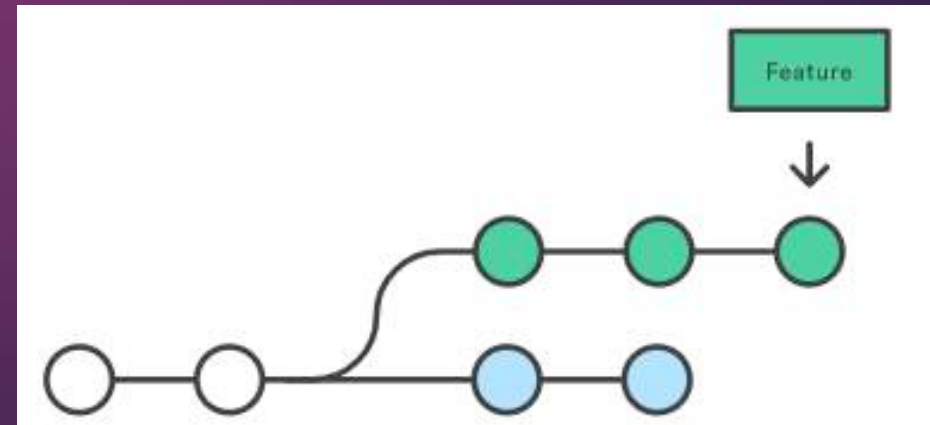
Local Operations

- ▶ When you modify files, you're updating your *working tree*
- ▶ When you add a file, you're moving it to the *staging area*, where it can be stored in a commit
- ▶ When you commit, Git essentially saves the files in the staging area in a commit object



Branches

- ▶ All a branch really is is a reference to a certain commit
- ▶ When you make a new commit on the a branch, Git automatically updates the branch to point to your new commit
- ▶ When you create a new branch, it points to the same commit that the branch you were just on did



Tags

- ▶ A tag is a reference to one particular commit
- ▶ Unlike branches, which update automatically, a tag will always point to the same commit
- ▶ Useful for marking versions

Head

- ▶ *Head* is a reference to the latest commit on the currently checked-out branch
- ▶ Many operations you do are implicitly on *head* without you knowing
- ▶ When you checkout a branch, all that's actually happening is Git changes what *head* points to



What's next?

More Commands and Tools

Stash

- ▶ A “pseudo-commit” stored in a special place
- ▶ For work-in-progress changes that aren't ready to be committed, but you need to checkout a different branch

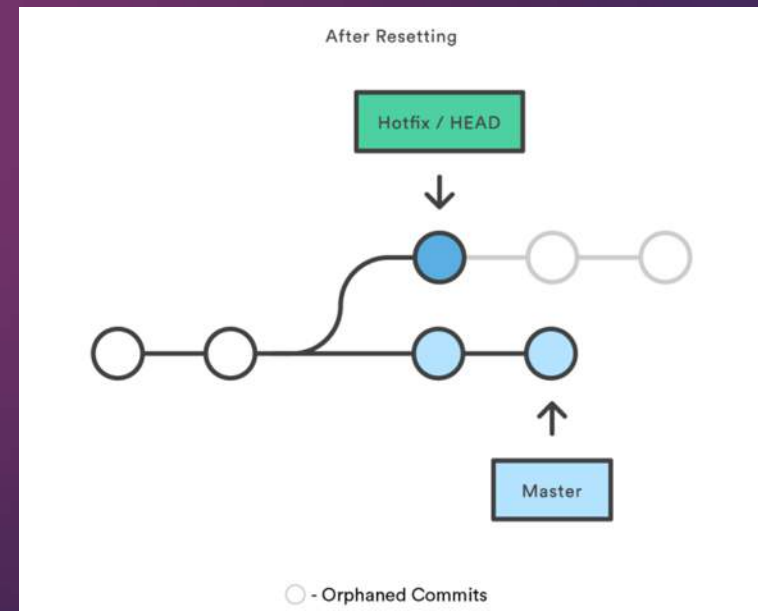
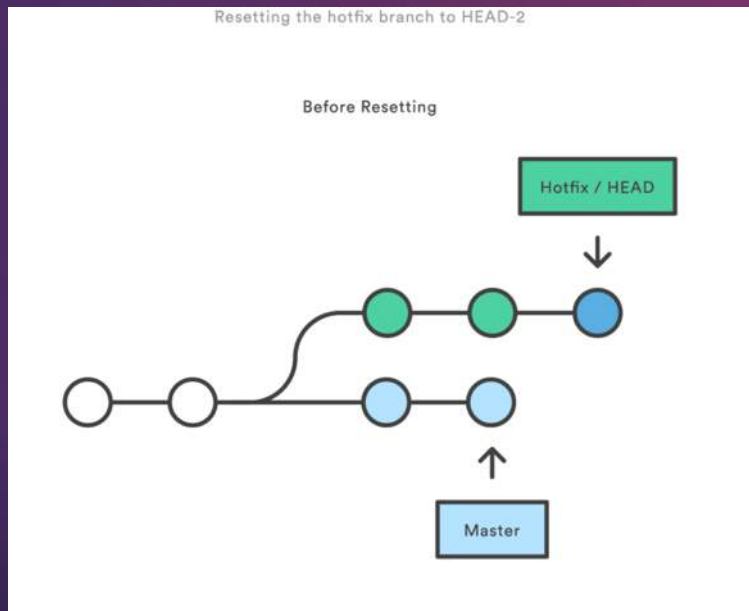
```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ vim feature.py
❯ git stash
Saved working directory and index state WIP on feature: b49374f Change feature
and add new file
❯ git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
❯ git checkout feature
Switched to branch 'feature'
❯ git stash pop
On branch feature
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   feature.py

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (04ba4ab88188b28122e6d23637712a6bf961e67f)
❯
```

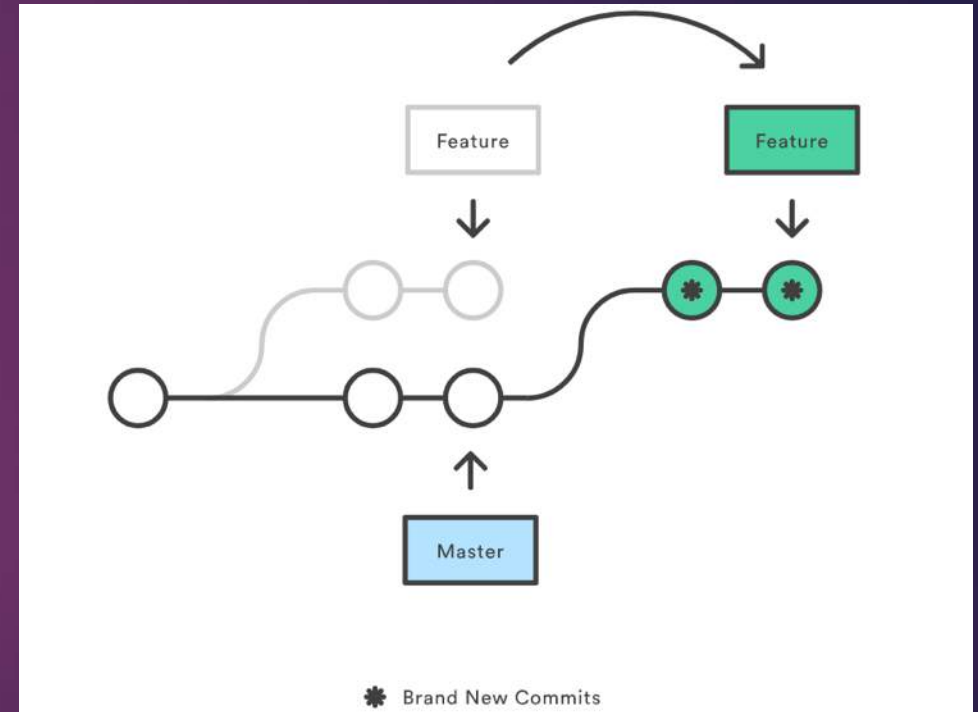
Reset

- ▶ Often used to reset changes and go back to an earlier commit, but...
- ▶ Fundamentally, *reset* just moves a branch head
- ▶ Has options to keep changes in the working tree or staging index, if desired



Rebase

- ▶ Reapply commits on a different parent
- ▶ Often used to preserve linear history
- ▶ Can also be used to drop, squash, and edit any commit
- ▶ Note: rebasing *changes history*, don't do this on a shared branch



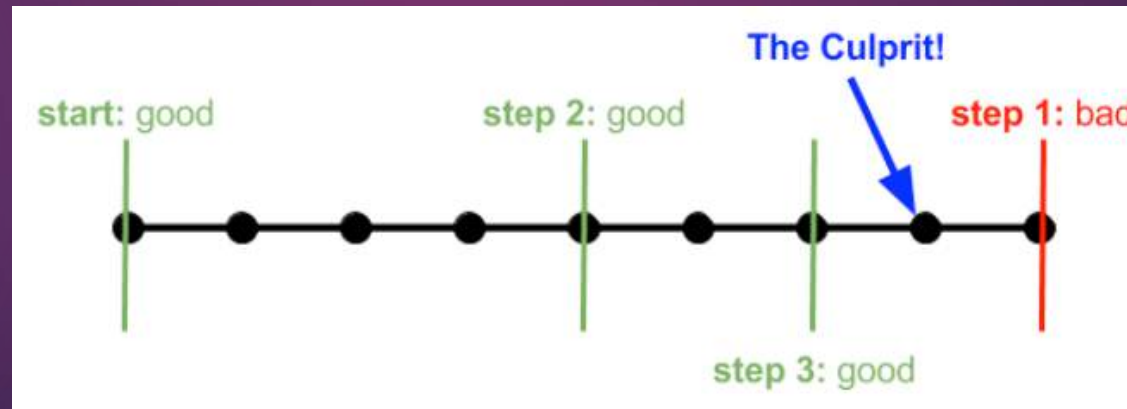
Reflog

- ▶ See a history of branches/commits you've checked out
- ▶ Especially useful if you accidentally reset too far and lose a commit, or if a rebase goes wrong

```
git reflog (less)
50c805e (HEAD -> master) HEAD@{0}: commit: Update start
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{1}: checkout: moving from feature to master
b49374f (feature) HEAD@{2}: checkout: moving from master to feature
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{3}: checkout: moving from feature to master
b49374f (feature) HEAD@{4}: reset: moving to HEAD
b49374f (feature) HEAD@{5}: checkout: moving from feature to feature
b49374f (feature) HEAD@{6}: checkout: moving from master to feature
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{7}: checkout: moving from bugfix to master
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{8}: reset: moving to HEAD
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{9}: rebase -i (finish): returning to refs/heads/bugfix
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{10}: rebase -i (start): checkout master
b76f3d3 HEAD@{11}: commit (merge): Merge origin/bugfix
d3e5ebc HEAD@{12}: checkout: moving from master to bugfix
d52c314 (origin/master, origin/bugfix, bugfix) HEAD@{13}: commit (merge): Merge origin/master
d3e5ebc HEAD@{14}: reset: moving to HEAD
d3e5ebc HEAD@{15}: checkout: moving from bugfix to master
:
```

Bisect

- ▶ Used to find the exact commit where a bug was introduced
- ▶ Start by specifying last known good commit
- ▶ Bisect will checkout commits in between and ask if they're good or bad
- ▶ At the end, bisect knows exactly which commit was the first bad one





Tips and tricks

to make your life easier

Pushing

- ▶ Set your default push action to current
- ▶ `git config push.default current`

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ git checkout -b feature
Switched to a new branch 'feature'
❯ vim feature.py
❯ git add .
❯ git commit -m "Implement feature"
[feature af9ecd2] Implement feature
1 file changed, 26 insertions(+)
create mode 100644 feature.py
❯ git config push.default current
❯ git push
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 321 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/BenChapmanKish/git-examples.git
 * [new branch]      feature -> feature
❯
```


Amending commits

- ▶ Fix typo in commit message
- ▶ Add file you forgot to stage
- ▶ `git commit --amend`

- ▶ Remember: commits can't truly be changed, this actually makes a new commit with your changes
- ▶ If you already pushed, you'll have to force push

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ vim feature.py
❯ git add .
❯ git commit -m "Refctro ftaeure"
[feature 1dd2126] Refctro ftaeure
1 file changed, 5 insertions(+), 9 deletions(-)
❯ git commit --amend -m "Refactor feature"
[feature 9afc27c] Refactor feature
Date: Tue Mar 27 23:57:42 2018 -0400
1 file changed, 5 insertions(+), 9 deletions(-)
❯
```

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ vim feature.py
❯ echo "new file" > new.txt
❯ git add feature.py
❯ git commit -m "Change feature and add new file"
[feature d4b1d6c] Change feature and add new file
1 file changed, 2 insertions(+), 4 deletions(-)
❯ git add new.txt
❯ git commit --amend --no-edit
[feature b49374f] Change feature and add new file
Date: Wed Mar 28 00:31:49 2018 -0400
2 files changed, 3 insertions(+), 4 deletions(-)
100644 new.txt
❯
```

Note the different hashes!

Merge conflicts

- ▶ Set your merge conflict style to diff3
- ▶ `git config --global merge.conflictstyle diff3`

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ git status
On branch master
Your branch is behind 'origin/master' by 1 commit, and can be fast-forwarded.
(use "git pull" to update your local branch)
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   fruits.txt

❯ git commit -m "Update fruits"
[master 7b6722d] Update fruits
 1 file changed, 1 insertion(+), 1 deletion(-)

❯ git pull
Auto-merging fruits.txt
CONFLICT (content): Merge conflict in fruits.txt
Recorded preimage for 'fruits.txt'
Automatic merge failed; fix conflicts and then commit the result.

❯ cat fruits.txt
apple

<<<<<< HEAD
mango
=====
orange
>>>>>> 668c6386da5fa928355a4c571ccfe29e6ab0ed66

pear
❯
```

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ git merge --abort
❯ git config merge.conflictstyle diff3
❯ git pull
Auto-merging fruits.txt
CONFLICT (content): Merge conflict in fruits.txt
Recorded preimage for 'fruits.txt'
Automatic merge failed; fix conflicts and then commit the result.

❯ cat fruits.txt
apple

<<<<<< HEAD
mango
|||||| merged common ancestors
radish
=====
orange
>>>>>> 668c6386da5fa928355a4c571ccfe29e6ab0ed66

pear
❯
```

Merge conflicts

- ▶ If someone else indented a bunch of lines and it's causing lots of conflicts
- ▶ `git merge feature -Xignore-all-space`

Diff algorithms

- ▶ Some diff algorithms can make much more sense than others
- ▶ `git diff --diff-algorithm=patience`
- ▶ Can also ignore whitespace changes with diff too
- ▶ `git diff -w`

Reuse recorded resolutions

- ▶ Never fix the same merge conflict twice!
- ▶ `git config --global rerere.enabled true`

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
└─ master 1 1 git config rerere.enabled true
└─ master 1 1 git pull
Auto-merging fruits.txt
CONFLICT (content): Merge conflict in fruits.txt
Recorded preimage for 'fruits.txt'
Automatic merge failed; fix conflicts and then commit the result.
└─ master | merge vim fruits.txt
└─ master | merge git add .
└─ master | merge git commit -m "Merge origin/master"
Recorded resolution for 'fruits.txt'.
[master d52c314] Merge origin/master
```

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
└─ master 2 git checkout bugfix
Switched to branch 'bugfix'
Your branch and 'origin/bugfix' have diverged,
and have 1 and 1 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)
└─ bugfix 1 1 git pull
Auto-merging fruits.txt
CONFLICT (content): Merge conflict in fruits.txt
Resolved 'fruits.txt' using previous resolution.
Automatic merge failed; fix conflicts and then commit the result.
└─ bugfix | merge git add .
└─ bugfix | merge git commit -m "Merge origin/bugfix"
[bugfix b76f3d3] Merge origin/bugfix
└─ bugfix 2
```

Handy shortcuts

- ▶ Checkout the previous branch you were on: `git checkout -`
- ▶ Reset n commits back on the current branch: `git reset @~ n`
- ▶ Add files and commit at the same time `git commit -a -m "message"`



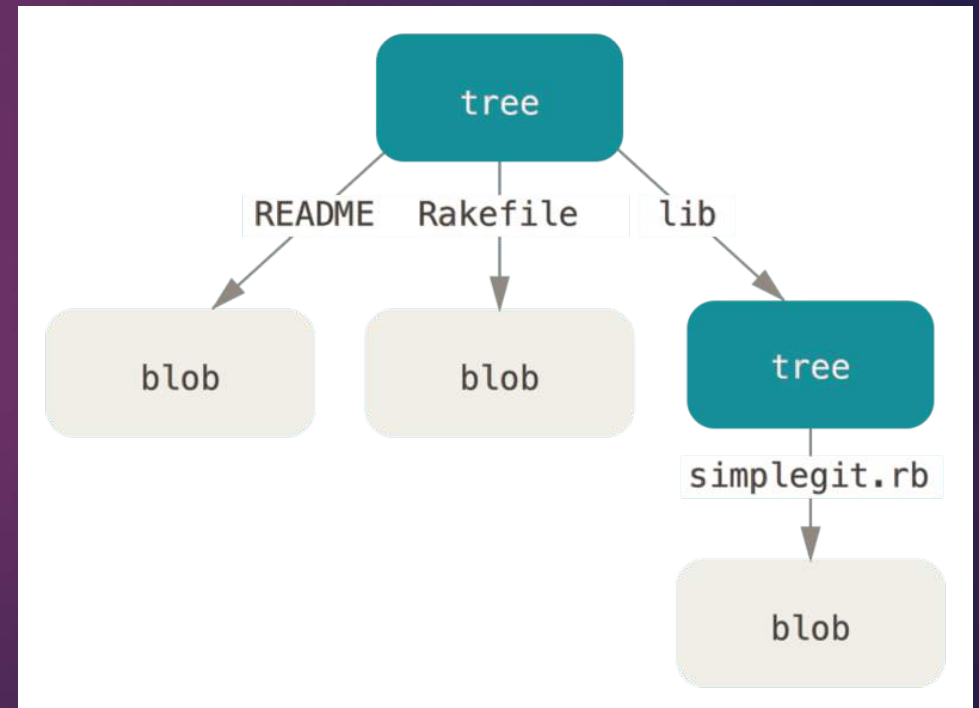
How does Git *really* work?

How Git *really* works

- ▶ Git is really a *content-addressable filesystem* with a VCS interface written on top of it
- ▶ Internally, Git has a key-value store of objects and their SHA-1 hashes (the hash is the key and the object is the value)
- ▶ These objects, among other Git internals, are stored in the `.git` directory at the root of every Git-controlled project

Git objects

- ▶ There are several types of objects that Git stores:
 - ▶ *Blob*: content; text/code/images/etc.
 - ▶ *Tree*: a collection of pointers to blobs and other trees, and names for each of these
 - ▶ *Commit*: A pointer to a tree, with metadata such as parent commits and a commit message



Plumbing and porcelain

- ▶ There are two kinds of Git commands:
 - ▶ The commands we use every day are called *porcelain* commands
 - ▶ Each of these actually uses low-level Git commands called *plumbing* commands
- ▶ Let's try using plumbing commands to do some basic Git operations!

Creating a blob

- ▶ Hashing content and storing the blob in the objects database
- ▶ No filename?

```
Ben@Bens-MacBook-Pro: ~/Documents/git-example (zsh)
❯ git hash-object -w --stdin
72da924ae664519ec5c1a30b74c8da2500e4aac8
❯ find .git/objects -type f
.git/objects/72/da924ae664519ec5c1a30b74c8da2500e4aac8
❯ git cat-file -t 72da924
blob
❯ git cat-file -p 72da924
This is some content
❯
```

Hashing the blob

- ▶ To get the hash, Git doesn't just hash the content
- ▶ It also prepends a header

```
python (Python)
>>> content = 'This is some content\n'
>>> header = 'blob %d\0' % len(content)
>>> store = header + content
>>> store
'blob 21\x00This is some content\n'
>>> sha1(store).hexdigest()
'72da924ae664519ec5c1a30b74c8da2500e4aac8'
>>> 
```

Reading a tree

- ▶ The tree stored in a commit on an actual project may look like this:

```
Ben@Bens-MacBook-Pro: ~/Documents/git-examples (zsh)
git cat-file -p master^{tree}
100644 blob 28e16b7041a9bd727aec7590414fec4eeacf6f49    about.txt
040000 tree 8375248684de1c069a9eb08e72a005ea62270fa8    assets
100644 blob b39e805ad672265be107dad3294ebaa4771b4ba1    fruits.txt
100644 blob 0e851fceed9476747ba4075540219953190eb0e8    server.py
100644 blob cfc6b2fc28b323b224041452843b18039dc7f132    start.py
100644 blob 29369f950d029a952db5cd21df772c6b27e14b0a    storage.db
git cat-file -p 8375248
100644 blob fd168a46742ae4d95535e8c669f7d270091f35b5    logo.png
```

Creating a tree

```
Ben@Bens-MacBook-Pro: ~/Documents/git-internals (zsh)
❯ echo 'This is version 1' | git hash-object -w --stdin
1b74696346d3ca52ae82f8fec4536488e05de302
❯ echo 'This is version 2' | git hash-object -w --stdin
ff91d982e271c96876c5dd5aa180944a77de574a
❯ git update-index --add --cacheinfo 100644 1b74696346d3ca5
2ae82f8fec4536488e05de302 file.txt
❯ git write-tree
a769d040896b27413c54edcaa0025bf00b10ad7a
❯ git cat-file -p a769d04
100644 blob 1b74696346d3ca52ae82f8fec4536488e05de302    file.txt
❯
```


Creating another tree

```
Ben@Bens-MacBook-Pro: ~/Documents/git-internals (zsh)
❯ echo 'This is a new file' > new.txt
❯ git update-index --add --cacheinfo 100644 ff91d982e271c96876c5dd5aa180944a77de574a file.txt
❯ git update-index --add new.txt
❯ git write-tree
9642c5ff94d0aae6aa40144c6e8c259d9e12a7b6
❯ git cat-file -p 9642c5f
100644 blob ff91d982e271c96876c5dd5aa180944a77de574a    file.txt
100644 blob 6dfa057f0d4a43d5a3025a9c14dea607de9e1dbb    new.txt
❯
```

```
Ben@Bens-MacBook-Pro: ~/Documents/git-internals (zsh)
❯ git read-tree --prefix=subdirectory 9642c5ff94d0aae6aa40144c6e8c259d9e12a7b6
❯ git write-tree
923bc6ec9703fa92c24d12037373ec38cf98803b
❯ git cat-file -p 923bc6e
100644 blob ff91d982e271c96876c5dd5aa180944a77de574a    file.txt
100644 blob 6dfa057f0d4a43d5a3025a9c14dea607de9e1dbb    new.txt
040000 tree 9642c5ff94d0aae6aa40144c6e8c259d9e12a7b6    subdirectory
❯
```

Creating a commit

```
Ben@Bens-MacBook-Pro: ~/Documents/git-internals (zsh)
❯ git commit-tree a769d04
3b39506538c7abefcfaefb8d01e7e5c8f9caa73a
❯ git cat-file -t 3b39506
commit
❯ git cat-file -p 3b39506
tree a769d040896b27413c54edcaa0025bf00b10ad7a
author Ben Chapman-Kish <ben.chapmankish@gmail.com> 1522301572 -0400
committer Ben Chapman-Kish <ben.chapmankish@gmail.com> 1522301572 -0400

First commit
❯
```

Creating a child commit

```
Ben@Bens-MacBook-Pro: ~/Documents/git-internals (zsh)
🐱 master ➤ echo 'Second commit' | git commit-tree 923bc6e -p 3b39506
72f6844e48761ffb17e2993079d2aa3855e312ad
🐱 master ➤ git cat-file -p 72f6844
tree 923bc6ec9703fa92c24d12037373ec38cf98803b
parent 3b39506538c7abefcfaefb8d01e7e5c8f9caa73a
author Ben Chapman-Kish <ben.chapmankish@gmail.com> 1522301752 -0400
committer Ben Chapman-Kish <ben.chapmankish@gmail.com> 1522301752 -0400

Second commit
🐱 master ➤ git log --stat 72f6844
```

```
git log --stat 72f6844 (less)
commit 72f6844e48761ffb17e2993079d2aa3855e312ad
Author: Ben Chapman-Kish <ben.chapmankish@gmail.com>
Date: Thu Mar 29 01:35:52 2018 -0400

    Second commit

    file.txt      | 2 +
    new.txt       | 1 +
    subdirectory/file.txt | 1 +
    subdirectory/new.txt | 1 +
    4 files changed, 4 insertions(+), 1 deletion(-)

commit 3b39506538c7abefcfaefb8d01e7e5c8f9caa73a
Author: Ben Chapman-Kish <ben.chapmankish@gmail.com>
Date: Thu Mar 29 01:32:52 2018 -0400

    First commit

    file.txt | 1 +
    1 file changed, 1 insertion(+)
(END)
```


There's so much more!

- ▶ Submodules
- ▶ References
- ▶ Packfiles
- ▶ Transfer protocols
- ▶ Garbage collection
- ▶ The refspec
- ▶ Git hooks



That's all for today!