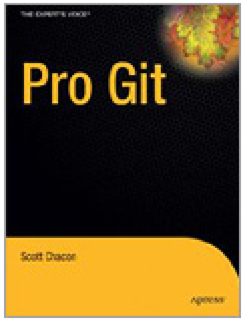
# Intermediate Git

A brownbag workshop at

by Seth House

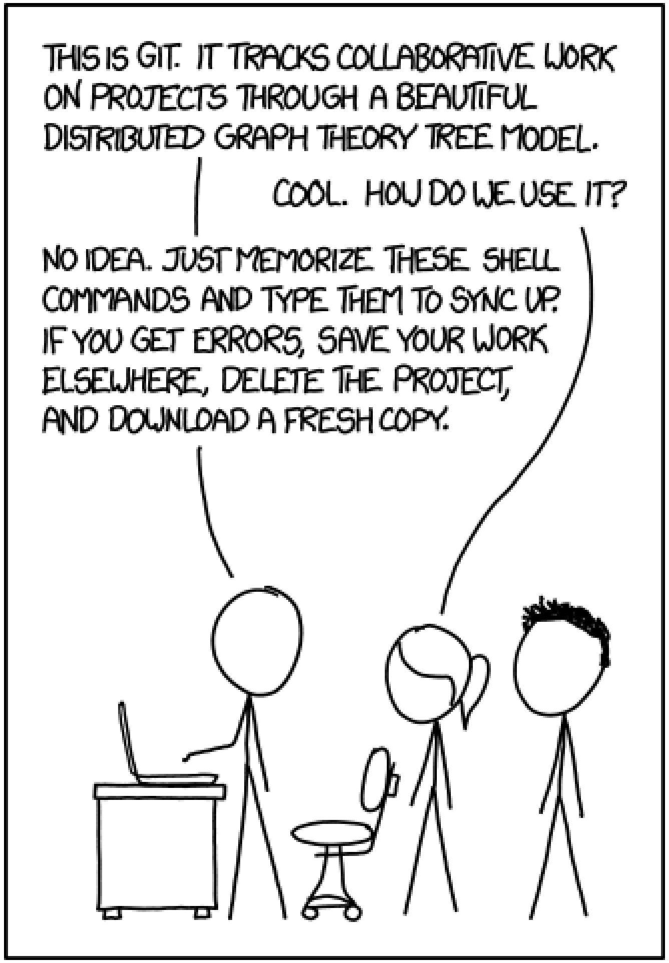
@whiteinge
seth@eseth.com

## Pro Git

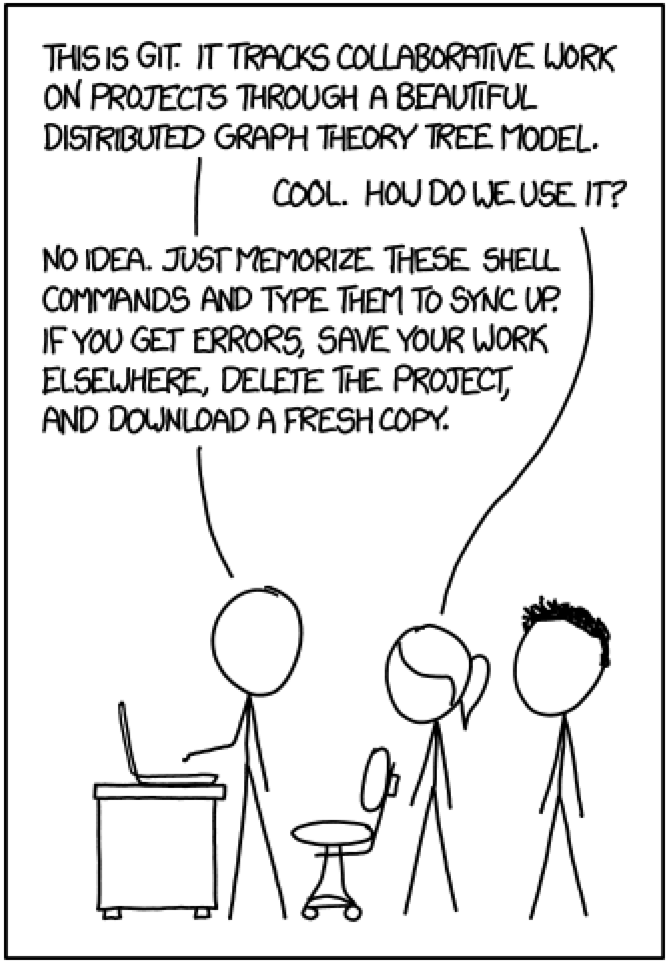


## Hands-on; Ask Questions!

# Hands-on; Ask Questions!


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

# Hands-on; Ask Questions!


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.

> If that doesn't fix it, git.txt contains the phone number of a friend of mine who understands git. Just wait through a few minutes of 'It's really pretty simple, just think of branches as...' and eventually you'll learn the commands that will fix everything.

## Just think of branches as...

## Just think of branches as...

- `find .git/refs`

## Just think of branches as...

- `find .git/refs`
- `cat .git/refs/heads/master`

## Just think of branches as...

- `find .git/refs`
- `cat .git/refs/heads/master`
- `less .git/packed-refs`

## Just think of branches as...

- `find .git/refs`
- `cat .git/refs/heads/master`
- `less .git/packed-refs`

"Lightweight" branches.

## Refs

- Refs are for humans. Git doesn't need them to function.
- Git only cares about the DAG (directed acyclic graph).

## Revisions

```
man gitrevisions
```

## Revisions

```
man gitrevisions
```

```
-  <sha1>, e.g. dae86e1950b1277e545cee180551750029cfe735, dae86e
   <describeOutput>, e.g. v1.7.4.2-679-g3bee7fb
-  <refname>, e.g. master, heads/master, refs/heads/master
        HEAD, FETCH_HEAD, ORIG_HEAD, MERGE_HEAD, CHERRY_PICK_HEAD
        refs/<refname>
        refs/tags/<refname>
        refs/heads/<refname>
        refs/remotes/<refname>
        refs/remotes/<refname>/HEAD
-  @
   [<refname>]@{<date>}, e.g. master@{yesterday}, HEAD@{5 minutes ago}
-  <refname>@{<n>}, e.g. master@{1}
   @{<n>}, e.g. @{1}
-  @{-<n>}, e.g. @{-1}
-  [<branchname>]@{upstream}, e.g. master@{upstream}, @{u}
   [<branchname>]@{push}, e.g. master@{push}, @{push}
   <rev>^[<n>], e.g. HEAD^, v1.5.1^0
-  <rev>~[<n>], e.g. HEAD~, master~3
   <rev>^{<type>}, e.g. v0.99.8^{commit}
   <rev>^{}, e.g. v0.99.8^{}
   <rev>^{/<text>}, e.g. HEAD^{/fix nasty bug}
-  :/<text>, e.g. :/fix nasty bug
   <rev>:<path>, e.g. HEAD:README, master:./README
   :[<n>:]<path>, e.g. :0:README, :README
```

## Ranges

```
   <rev>
   ^<rev>
-  <rev1>..<rev2>, e.g. @{u}..HEAD, HEAD..@{u}, @{u}.., ..@{u}
-  <rev1>...<rev2>, e.g. @{u}...HEAD, HEAD...@{u}, @{u}..., ...@{u}
   <rev>^@, e.g. HEAD^@
   <rev>^!, e.g. HEAD^!
   <rev>^-<n>, e.g. HEAD^-, HEAD^-2
```

## Local and "remote" refs

- Local and "remote" refs are both stored in the `.git` directory.

## Local and "remote" refs

- Local and "remote" refs are both stored in the `.git` directory.
- A branch can have an "upstream" ref association.

# Local and "remote" refs

- Local and "remote" refs are both stored in the `.git` directory.
- A branch can have an "upstream" ref association.
- `fetch` updates the local DAG.
  `pull` updates the local DAG *and* moves ref pointers.

## Reflog

- `git reflog --date=relative`
- Any objects on the graph without a ref will be garbage collected.
- The reflog counts as a reference!
- Persists for 90-days (default).

## Reflog

- `git reflog --date=relative`
- Any objects on the graph without a ref will be garbage collected.
- The reflog counts as a reference!
- Persists for 90-days (default).
- If a change is saved in a commit object, it is *safe* and can be recovered.

## Reflog

- `git reflog --date=relative`
- Any objects on the graph without a ref will be garbage collected.
- The reflog counts as a reference!
- Persists for 90-days (default).
- If a change is saved in a commit object, it is *safe* and can be recovered. (Stashes are *not* commit objects.)

## Rebase or merge?

## Rebase or merge?

- Commits communicate *intent* to your teammates.

## Rebase or merge?

- Commits communicate *intent* to your teammates.
- What are you trying to communicate with a given branch or merge?

## Rebase or merge?

- Commits communicate *intent* to your teammates.
- What are you trying to communicate with a given branch or merge?
- E.g.:
  - A clean, linear history of a feature addition or bug fix?
  - Record of when and who updated a branch?

## Rebase or merge?

- Commits communicate *intent* to your teammates.
- What are you trying to communicate with a given branch or merge?
- E.g.:
    - A clean, linear history of a feature addition or bug fix?
    - Record of when and who updated a branch?
- It's too confusing to rebase a shared branch.

## Rebase

- `git rebase -i`
- `git commit --fixup` (`git add -p`)
- `git commit --amend`
- `git rebase --onto`

## Rebase

- `git rebase -i`
- `git commit --fixup` (`git add -p`)
- `git commit --amend`
- `git rebase --onto`

(Future presentation?)

## Merge strategy (default: "recursive")

`man git-merge-base`, `man git-merge-file`, `man git-merge`, `man git-diff`

## Merge strategy (default: "recursive")

`man git-merge-base`, `man git-merge-file`, `man git-merge`, `man git-diff`

```
      A---B---C topic
     /
D---E---F---G master
```

# Merge strategy (default: "recursive")

`man git-merge-base`, `man git-merge-file`, `man git-merge`, `man git-diff`

```
      A---B---C topic
     /
D---E---F---G master
```

1. Find the common ancestor, the merge base, of both branches. If the merge base is, itself, a merge then *recurse* and follow the ancestry farther up.

## Merge strategy (default: "recursive")

`man git-merge-base`, `man git-merge-file`, `man git-merge`, `man git-diff`

```
      A---B---C topic
     /
D---E---F---G master
```

1. Find the common ancestor, the merge base, of both branches. If the merge base is, itself, a merge then *recurse* and follow the ancestry farther up.

2. Replay changes made on `topic` (including file renames) since it diverged (`E`). Record the result in a new commit with the two parent SHAs and a log message from the user.

## Merge strategy (default: "recursive")

`man git-merge-base`, `man git-merge-file`, `man git-merge`, `man git-diff`

```
      A---B---C topic
     /
D---E---F---G master
```

1. Find the common ancestor, the merge base, of both branches. If the merge base is, itself, a merge then *recurse* and follow the ancestry farther up.

2. Replay changes made on `topic` (including file renames) since it diverged (`E`). Record the result in a new commit with the two parent SHAs and a log message from the user.

3. On each replay, each file in working tree is diffed and updated:

    - Start with common ancestor's version of the file.
    - Non-overlapping areas are incorporated verbatim.
    - Changes made to both sides of an area will be a conflict and Git wraps both sides in conflict markers so the user can choose.
    - Diff overlap detection can be tuned by changing the diff algorithm used (patience, minimal, histogram, myers).

## Merge conflicts

## Merge conflicts

- Differences in the surrounding diff *context*.
- Whitespace differences.
- Conflicting changes.

## Conflict resolution

```
git merge -X ours feature1
git merge -X theirs feature1
```

## Conflict resolution

```
git merge -X ours feature1
git merge -X theirs feature1
```

Be sure!

## Conflict markers

## Conflict markers

```
<<<<<<< HEAD
twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
all mimsy were the borogoves,
And the mome raths outgrabe.
=======
'Twas brillig, and the slithy toves
Did gyre and gimble in the wabe:
All mimsy were the borogroves
And the mome raths outgabe.
>>>>>>> branchA

"Beware the Jabberwock, my son!
The jaws that bite, the claws that catch!
Beware the Jubjub bird, and shun
The frumious Bandersnatch!"
```

## Conflict history

Advanced Merging (Pro Git book)

## Conflict history

Advanced Merging (Pro Git book)

- Investigate relevant changes:

```
git diff --ours
git diff --theirs
```

## Conflict history

Advanced Merging (Pro Git book)

- Investigate relevant changes:

```
git diff --ours
git diff --theirs
```

- Look at each version of the conflicted file (1 Base, 2 Ours, 3 Theirs):

```
git show :1:poem.txt
git show :2:poem.txt
git show :3:poem.txt
# (or)
git ls-files -u
```

## Conflict history

Advanced Merging (Pro Git book)

- Investigate relevant changes:

```
git diff --ours
git diff --theirs
```

- Look at each version of the conflicted file (1 Base, 2 Ours, 3 Theirs):

```
git show :1:poem.txt
git show :2:poem.txt
git show :3:poem.txt
# (or)
git ls-files -u
```

- Follow the file history:

```
git log --oneline --left-right HEAD...MERGE_HEAD
git log --oneline --left-right --merge
git log --oneline --left-right --merge -p
git log --oneline --left-right --merge -p -- poem.txt
```

## mergetools

# mergetools

- `LOCAL` - What the file looks like on your branch (before merge!).
- `REMOTE` - What the file looks like on the other branch (before merge!).
- `BASE` - What the file looks like from before your branch and the other branch diverged. (The "merge base" or the "common ancestor".)

## mergetools

- `LOCAL` - What the file looks like on your branch (before merge!).
- `REMOTE` - What the file looks like on the other branch (before merge!).
- `BASE` - What the file looks like from before your branch and the other branch diverged. (The "merge base" or the "common ancestor".)
- `MERGED` - Everything Git was able to resolve automatically and also everything that it was not able to resolve.

## mergetools

- `LOCAL` - What the file looks like on your branch (before merge!).
- `REMOTE` - What the file looks like on the other branch (before merge!).
- `BASE` - What the file looks like from before your branch and the other branch diverged.
  (The "merge base" or the "common ancestor".)
- `MERGED` - Everything Git was able to resolve automatically and also everything that it was not able to resolve.
  (A two-way diff.)

## Advanced Git

- Commit objects -> tree objects -> blob objects.

```
git cat-file -p HEAD
```

- Git packs and (re-)calculating diffs between commits.
- Non-standard refs.
- Refspecs.
- Transfer protocol.
- Maintenance and data recovery.