git 201
+ how to play well with others
Stacy Watts

Devops Days Portland
Tuesday, September 10, 2019

Conferences of all stripes take a lot of time and effort.

Thank you volunteers!

Code of Conduct

https://devopsdays.org/events/2019-portland/conduct

Open Spaces

https://devopsdays.org/open-space-format/

Whoever comes is the right people

Whatever happens happens

Whenever it starts is the right time

When it’s over it’s over

The law of mobility

Bring your best self

A bit about me

9102019

I may be a minor math nerd. Today in the US is a palindrome.

Since there is no "0" month, the next such date will be January 20, 2021, or 1202021.

I'm very much looking forward to that date as well!

(Work)

Senior MSS Platform Engineer

Bluevoyant

(Volunteer)

Women Who Code Portland

Devops Study Night co-lead
Security Study Night co-lead

Aurora Chorus

Soprano 1

About Women Who Code

We are a global nonprofit dedicated to inspiring women to excel in technology careers. Our events offer study groups, technical workshops, hackathons, networking events, panel discussions, lightning talks, and social events featuring influential tech industry experts, innovators, and investors. We help you build the skills you need to raise your professional profile and achieve greater career success. Current and aspiring coders are welcome.

Git 101 review

The work of today is the history of tomorrow, and we are its makers

attributed to a letter written by Juliette Low, founder of the Girl Scouts, in 1925

Another way to look at this

Code is legacy when written.

me[1], 9102019

[1] This isn't a new idea, I didn't invent it. But it doesn't appear to be on the internet yet, so now it is. :)

Link to git in a nutshell:

https://git-scm.com/book/en/v2/Getting-Started-What-is-Git%3F

Things to remember:

- every change to the remote repository is in every local copy checked out
- you can always go back in time, including the beginning of the repo

config	**merge**	**init**	**clone**
add	status	diff	commit
reset	rm	mv	branch
checkout	merge	log	stash
tag	fetch	pull	push
remote	show	shortlog	

git 201

git 201

grep	merge	rebase
cherry-pick	revert	blame

grep

the most underused tool on the planet

Examples from the documentation:

```
$ man git-grep

...

git grep 'time_t' -- '*.[ch]'
    Looks for time_t in all tracked .c and .h files in the working directory and its subdirectories.

git grep -e '#define' --and \( -e MAX_PATH -e PATH_MAX \)
    Looks for a line that has #define and either MAX_PATH or PATH_MAX.

git grep --all-match -e NODE -e Unexpected
    Looks for a line that has NODE or Unexpected in files that have lines that match both.

git grep solution -- :^Documentation
    Looks for solution, excluding files in Documentation
```

Some I use every day:

```
$ man git-grep

...

$ git grep -i foo
    Looks for foo case insensitive in the current repository.  Super useful when someone asks if you know where
    feature "foo" is or variable "foo" is defined.

$ for i in $( find ~/repos -name ".git" | sed 's/....$//'); do cd $i; git grep -i foo $i; done
    Chaining a few cli ideas together - loops over all found git repos in the directory you specified for the find command.
    Similar to the above, look through all repos found for the needle "foo" in your haystacks.

    While this is needle in a haystack, multiply the number of devs in your org by the number of workdays in one year.
    Then go back to the idea of git history and trying to remember when something changes and you get something like:

$ git grep -i foo $(git rev-list --all)
    This may bomb out if your Argument List is too long, too many commits, etc.  At that point look to something like
    the above mixed with xargs.  The command is effectively looking in every possible point in the repo's history for foo
    Personally I like git log -p and the search using less's search.  Seems faster, but I can also see the above being useful.

    You can learn a great deal about the codebases at a new job with git grep.
```

merge

conflicts

```
$ man git-merge

...

EXAMPLES
       o   Merge branches fixes and enhancements on top of the current branch, making an octopus merge:

               $ git merge fixes enhancements

       o   Merge branch obsolete into the current branch, using ours merge strategy:

               $ git merge -s ours obsolete

       o   Merge branch maint into the current branch, but do not make a new commit automatically:

               $ git merge --no-commit maint
$ man git-merge
...
MERGE STRATEGIES
       The merge mechanism (git merge and git pull commands) allows the backend merge strategies to be chosen with -s option. Some strategies can also take their
       own options, which can be passed by giving -X<option> arguments to git merge and/or git pull.
...
       recursive
           This can only resolve two heads using a 3-way merge algorithm. When there is more than one common ancestor that can be used for 3-way merge, it creates
           a merged tree of the common ancestors and uses that as the reference tree for the 3-way merge. This has been reported to result in fewer merge
           conflicts without causing mismerges by tests done on actual merge commits taken from Linux 2.6 kernel development history. Additionally this can detect
           and handle merges involving renames, but currently cannot make use of detected copies. This is the default merge strategy when pulling or merging one
           branch.

           The recursive strategy can take the following options:

           ours
               This option forces conflicting hunks to be auto-resolved cleanly by favoring our version. Changes from the other tree that do not conflict with our
               side are reflected to the merge result. For a binary file, the entire contents are taken from our side.

               This should not be confused with the ours merge strategy, which does not even look at what the other tree contains at all. It discards everything
               the other tree did, declaring our history contains all that happened in it.

           theirs
               This is the opposite of ours; note that, unlike ours, there is no theirs merge strategy to confuse this merge option with.

           patience
               With this option, merge-recursive spends a little extra time to avoid mismerges that sometimes occur due to unimportant matching lines (e.g.,
               braces from distinct functions). Use this when the branches to be merged have diverged wildly. See also git-diff(1) --patience.
...
           ignore-space-change, ignore-all-space, ignore-space-at-eol, ignore-cr-at-eol
               Treats lines with the indicated type of whitespace change as unchanged for the sake of a three-way merge. Whitespace changes mixed with other
               changes to a line are not ignored. See also git-diff(1) -b, -w, --ignore-space-at-eol, and --ignore-cr-at-eol.
...
       octopus
           This resolves cases with more than two heads, but refuses to do a complex merge that needs manual resolution. It is primarily meant to be used for
           bundling topic branch heads together. This is the default merge strategy when pulling or merging more than one branch.

       ours
           This resolves any number of heads, but the resulting tree of the merge is always that of the current branch head, effectively ignoring all changes from
           all other branches. It is meant to be used to supersede old development history of side branches. Note that this is different from the -Xours option to
           the recursive merge strategy.

       subtree
           This is a modified recursive strategy. When merging trees A and B, if B corresponds to a subtree of A, B is first adjusted to match the tree structure
           of A, instead of reading the trees at the same level. This adjustment is also done to the common ancestor tree.

       With the strategies that use 3-way merge (including the default, recursive), if a change is made on both branches, but later reverted on one of the
       branches, that change will be present in the merged result; some people find this behavior confusing. It occurs because only the heads and the merge base
       are considered when performing a merge, not the individual commits. The merge algorithm therefore considers the reverted change as no change at all, and
       substitutes the changed version instead.
```

Dealing with merge conflicts is one of the halmarks of coding on a team. No matter how hard you try, you will overwrite, write, whitespace change, refactor, or rename something that someone else also changed the same week or even the same hour.

Let's practice undoing a merge.

```
$ git status
# On branch great-feature
# You have unmerged paths.
#   (fix conflicts and run "git commit")
#
# Unmerged paths:
#   (use "git add <file>..." to mark resolution)
#
#       both modified:   main_page.html
#
no changes added to commit (use "git add" and/or "git commit -a")

$ git merge --abort
    Abort the process, think about thigs, check in what that teammate!  speaking of:

$ git blame main_page.html
...
6bf0899f 2019_09_10_git_201_how_to_play_well_with_others.md   (Stacy Watts       2019-09-01 21:04:32 -0700  30) ## Senior MSS Platform Engineer
6bf0899f 2019_09_10_git_201_how_to_play_well_with_others.md   (Stacy Watts       2019-09-01 21:04:32 -0700  31) # Bluevoyant
...
    Find out who you need to get in touch with, or if they're in another timezone possibly which ticket they were working for clues.

$ git reset --hard
    This is in case you realize you got through the merge, made an error, and want to go back to the commit before the merge.
    Effectively you start over, but don't worry, you still have your changes there to merge or not to merge!
    There is much much more to git-reset.  See the man page or the git-scm.com book for further good examples.
```

```
$ git merge --abort
$ git merge -Xignore-space-change whitespace
    You find the conflict is all whitespace, abort, go for this strategy and all is well.

$ git checkout --conflict=diff3 file.txt
    diff3 give you one more piece of data - diff only gives you ours and theirs.  diff3 tells you what the common
    ancestor had in it.  Often I can find the answer to what changed more easily this way with a more explicit diff3 tool.


def hello
<<<<<<< ours
  puts 'hola world'
||||||| base
  puts 'hello world'
=======
  puts 'hello mundo'
>>>>>>> theirs


$ git merge --abort
$ git merge --strategy-option=theirs
    You know theirs are the changes you should use, you saw the diff.  So why look at it?  Let git do the merge for you.

$ git merge --abort
$ git merge --strategy-option=ours
    You know yours are the changes you should use, you saw the diff.  So why look at it?  Let git do the merge for you.
```

rebase

for when history diverges

```
$ man git-rebase

NAME
       git-rebase - Reapply commits on top of another base tip

SYNOPSIS
       git rebase [-i | --interactive] [<options>] [--exec <cmd>] [--onto <newbase>]
               [<upstream> [<branch>]]
       git rebase [-i | --interactive] [<options>] [--exec <cmd>] [--onto <newbase>]
               --root [<branch>]
       git rebase --continue | --skip | --abort | --quit | --edit-todo | --show-current-patch
...

       Assume the following history exists and the current branch is "topic":

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

       From this point, the result of either of the following commands:

           git rebase master
           git rebase master topic

       would be:

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

...

more graph examples in the man page
```

```
$ git checkout great-feature
$ git rebase master
First, rewinding head to replay your work on top of it...
Applying: my great commit message
$ git checkout master
$ git merge great-feature

    What about  multiple branches?

$ git checkout great-feature
$ git rebase --onto master their-branch great-feature
    What what was that?  Pretend like you played out all the commits since your branch diverged from
    their-branch onto master.  That easy.
```

interactive rebase

```
$ git rebase -i HEAD~6
...
pick 039dafe add Git 201 + how to play well with others talk md file
pick 03b1ebf rename
pick 6bf0899 more changes
pick 0b04410 more
pick 8fbe3a3 final changes
pick a967ba8 6

# Rebase f979d3f..a967ba8 onto f979d3f (6 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
```

```
pick 039dafe add Git 201 + how to play well with others talk md file
s 03b1ebf rename
s 6bf0899 more changes
s 0b04410 more
s 8fbe3a3 final changes
s a967ba8 6

# Rebase f979d3f..a967ba8 onto f979d3f (6 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup <commit> = like "squash", but discard this commit's log message
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
# .       create a merge commit using the original merge commit's
# .       message (or the oneline, if no original merge commit was
# .       specified). Use -c <commit> to reword the commit message.
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# If you remove a line here THAT COMMIT WILL BE LOST.
#
# However, if you remove everything, the rebase will be aborted.
#
# Note that empty commits are commented out
# This is a combination of 6 commits.
# This is the 1st commit message:
add Git 201 + how to play well with others talk md file
# This is the commit message #2:
# This is the commit message #3:
# This is the commit message #4:
# This is the commit message #5:
# This is the commit message #6:
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Sun Sep 8 19:31:06 2019 -0700
#
# interactive rebase in progress; onto f979d3f
# Last commands done (6 commands done):
#    squash 8fbe3a3 final changes
#    squash a967ba8 6
# No commands remaining.
# You are currently rebasing branch 'git-201' on 'f979d3f'.
#
# Changes to be committed:
# new file:   2019_09_10_git_201_how_to_play_well_with_others.md
#
```

save again and you're set.

cherry-pick

```
$ man git-cherry-pick
...
NAME
       git-cherry-pick - Apply the changes introduced by some existing commits

SYNOPSIS
       git cherry-pick [--edit] [-n] [-m parent-number] [-s] [-x] [--ff]
                         [-S[<keyid>]] <commit>...
       git cherry-pick --continue
       git cherry-pick --quit
       git cherry-pick --abort

DESCRIPTION
       Given one or more existing commits, apply the change each one introduces, recording a new commit for each. This requires your working tree to be clean (no
       modifications from the HEAD commit).
...
EXAMPLES
       git cherry-pick master
           Apply the change introduced by the commit at the tip of the master branch and create a new commit with this change.

       git cherry-pick ..master, git cherry-pick ^HEAD master
           Apply the changes introduced by all commits that are ancestors of master but not of HEAD to produce new commits.

       git cherry-pick maint next ^master, git cherry-pick maint master..next
           Apply the changes introduced by all commits that are ancestors of maint or next, but not master or any of its ancestors. Note that the latter does not
           mean maint and everything between master and next; specifically, maint will not be used if it is included in master.

       git cherry-pick master~4 master~2
           Apply the changes introduced by the fifth and third last commits pointed to by master and create 2 new commits with these changes.

       git cherry-pick -n master~1 next
           Apply to the working tree and the index the changes introduced by the second last commit pointed to by master and by the last commit pointed to by
           next, but do not create any commit with these changes.

       git cherry-pick --ff ..next
           If history is linear and HEAD is an ancestor of next, update the working tree and advance the HEAD pointer to match next. Otherwise, apply the changes
           introduced by those commits that are in next but not HEAD to the current branch, creating a new commit for each new change.

       git rev-list --reverse master -- README | git cherry-pick -n --stdin
           Apply the changes introduced by all commits on the master branch that touched README to the working tree and index, so the result can be inspected and
           made into a single new commit if suitable.
```

revert

```
$ man git-revert
...

NAME
       git-revert - Revert some existing commits

SYNOPSIS
       git revert [--[no-]edit] [-n] [-m parent-number] [-s] [-S[<keyid>]] <commit>...
       git revert --continue
       git revert --quit
       git revert --abort

DESCRIPTION
       Given one or more existing commits, revert the changes that the related patches introduce, and record some new commits that record them. This requires your
       working tree to be clean (no modifications from the HEAD commit).

       Note: git revert is used to record some new commits to reverse the effect of some earlier commits (often only a faulty one). If you want to throw away all
       uncommitted changes in your working directory, you should see git-reset(1), particularly the --hard option. If you want to extract specific files as they
       were in another commit, you should see git-checkout(1), specifically the git checkout <commit> -- <filename> syntax. Take care with these alternatives as
       both will discard uncommitted changes in your working directory.
...

EXAMPLES
       git revert HEAD~3
           Revert the changes specified by the fourth last commit in HEAD and create a new commit with the reverted changes.

       git revert -n master~5..master~2
           Revert the changes done by commits from the fifth last commit in master (included) to the third last commit in master (included), but do not create any
           commit with the reverted changes. The revert only modifies the working tree and the index.
```

blame

```
$ man git-blame
...
NAME
       git-blame - Show what revision and author last modified each line of a file

SYNOPSIS
       git blame [-c] [-b] [-l] [--root] [-t] [-f] [-n] [-s] [-e] [-p] [-w] [--incremental]
                    [-L <range>] [-S <revs-file>] [-M] [-C] [-C] [-C] [--since=<date>]
                    [--progress] [--abbrev=<n>] [<rev> | --contents <file> | --reverse <rev>..<rev>]
                    [--] <file>

DESCRIPTION
       Annotates each line in the given file with information from the revision which last modified the line. Optionally, start annotating from the given
       revision.

       When specified one or more times, -L restricts annotation to the requested lines.

       The origin of lines is automatically followed across whole-file renames (currently there is no option to turn the rename-following off). To follow lines
       moved from one file to another, or to follow lines that were copied and pasted from another file, etc., see the -C and -M options.

       The report does not tell you anything about lines which have been deleted or replaced; you need to use a tool such as git diff or the "pickaxe" interface
       briefly mentioned in the following paragraph.

       Apart from supporting file annotation, Git also supports searching the development history for when a code snippet occurred in a change. This makes it
       possible to track when a code snippet was added to a file, moved or copied between files, and eventually deleted or replaced. It works by searching for a
       text string in the diff. A small example of the pickaxe interface that searches for blame_usage:

           $ git log --pretty=oneline -S'blame_usage'
           5040f17eba15504bad66b14a645bddd9b015ebb7 blame -S <ancestry-file>
           ea4c7f9bf69e781dd0cd88d2bccb2bf5cc15c9a7 git-blame: Make the output

OPTIONS
       -b
           Show blank SHA-1 for boundary commits. This can also be controlled via the blame.blankboundary config option.

       --root
           Do not treat root commits as boundaries. This can also be controlled via the blame.showRoot config option.

       --show-stats
           Include additional statistics at the end of blame output.

       -L <start>,<end>, -L :<funcname>
           Annotate only the given line range. May be specified multiple times. Overlapping ranges are allowed.
```

```
$ git blame botocore/data/config/2014-11-12/service-2.json
...
c258624f2 botocore/data/config/2014-11-12/service-2.json  (AWS                       2016-07-21 22:58:25 +0000  117)       ],
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  118)       "documentation":"<p>Deletes the evaluation results for the
 specified AWS Config rule. You can specify one AWS Config rule per request. After you delete the evaluation results, you can call the <a>StartConfigRulesEvaluation</a> API to start
 evaluating your AWS resources against the rule.</p>"
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  119)     },
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  120)     "DeleteOrganizationConfigRule":{
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  121)       "name":"DeleteOrganizationConfigRule",
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  122)       "http":{
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  123)         "method":"POST",
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  124)         "requestUri":"/"
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  125)       },
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  126)       "input":{"shape":"DeleteOrganizationConfigRuleRequest"},
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  127)       "errors":[
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  128)         {"shape":"NoSuchOrganizationConfigRuleException"},
c7614c90b botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-17 18:08:17 +0000  129)         {"shape":"ResourceInUseException"},
c7614c90b botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-17 18:08:17 +0000  130)         {"shape":"OrganizationAccessDeniedException"}
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  131)       ],
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  132)       "documentation":"<p>Deletes the specified organization config rule and all of its evaluation results from all member accounts\
 in that organization. Only a master account can delete an organization config rule.</p> <p>AWS Config sets the state of a rule to DELETE_IN_PROGRESS until the deletion is complete. You cannot update a rule while it is in this state.</p>"
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  133)     },
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  134)     "DeletePendingAggregationRequest":{
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  135)       "name":"DeletePendingAggregationRequest",
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  136)       "http":{
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  137)         "method":"POST",
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  138)         "requestUri":"/"
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  139)       },
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  140)       "input":{"shape":"DeletePendingAggregationRequestRequest"},
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  141)       "errors":[
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  142)         {"shape":"InvalidParameterValueException"}
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  143)       ],
861992908 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-04-04 11:39:42 -0700  144)       "documentation":"<p>Deletes pending authorization requests for a specified aggregator account in a specified region.</p>"
c258624f2 botocore/data/config/2014-11-12/service-2.json  (AWS                       2016-07-21 22:58:25 +0000  145)     },
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  146)     "DeleteRemediationConfiguration":{
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  147)       "name":"DeleteRemediationConfiguration",
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  148)       "http":{
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  149)         "method":"POST",
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  150)         "requestUri":"/"
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  151)       },
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  152)       "input":{"shape":"DeleteRemediationConfigurationRequest"},
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  153)       "output":{"shape":"DeleteRemediationConfigurationResponse"},
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  154)       "errors":[
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  155)         {"shape":"NoSuchRemediationConfigurationException"},
6a525419c botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-07-09 18:29:37 +0000  156)         {"shape":"RemediationInProgressException"}
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  157)       ],
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  158)       "documentation":"<p>Deletes the remediation configuration.</p>"
a7bbfe904 botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-03-13 11:02:24 -0700  159)     },
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  160)     "DeleteRemediationExceptions":{
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  161)       "name":"DeleteRemediationExceptions",
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  162)       "http":{
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  163)         "method":"POST",
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  164)         "requestUri":"/"
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  165)       },
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  166)       "input":{"shape":"DeleteRemediationExceptionsRequest"},
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  167)       "output":{"shape":"DeleteRemediationExceptionsResponse"},
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  168)       "errors":[
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  169)         {"shape":"NoSuchRemediationExceptionException"}
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  170)       ],
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  171)       "documentation":"<p>Deletes one or more remediation exceptions mentioned in the resource keys.</p>"
cef35f5fc botocore/data/config/2014-11-12/service-2.json  (aws-sdk-python-automation 2019-09-05 18:19:50 +0000  172)     },
7e17b2d91 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-05-25 14:29:56 -0700  173)     "DeleteRetentionConfiguration":{
7e17b2d91 botocore/data/config/2014-11-12/service-2.json  (awstools                  2018-05-25 14:29:56 -0700  174)       "name":"DeleteRetentionConfiguration",
...
```

plumbing commands

git 301 briefly into the matrix of internals

https://git-scm.com/docs

```
$ git ls-files
2019_09_10_git_201_how_to_play_well_with_others.md
README.md

$ git ls-files --stage
100644 58cc487a7831417541268478ac01840db0c398f5 0   2019_09_10_git_201_how_to_play_well_with_others.md
100644 670ecbed8112164bb7d6f75170a749c8775a1441 0   README.md

$ git write-tree
835435fbebfe4f259c8016c2f872b01588800790

$ git ls-tree 8354
100644 blob 58cc487a7831417541268478ac01840db0c398f5    2019_09_10_git_201_how_to_play_well_with_others.md
100644 blob 670ecbed8112164bb7d6f75170a749c8775a1441    README.md
```

how to play well with others

Cultures

Much of written language is culture. This goes for code as much as it does for civilizations.

Culture can be: American, agile, waterfall, White, Native, Black, Ruby, Python, Mac, Linux, Windows, BSD, Mob programming, XP (extreme programming), Indian, Hungarian, Open Source, FOSS, gendered .... you get the idea.

Know or learn your audience for effective collaboration on code. Their culture will be different from yours. Learn to love and respect that.

Expect the unexpected

How to contribute to open-source

- First, find a project
- Second, find the contribution guidelines
- Third find the irc, or whichever community channel they discuss the code base in. Ask any questions on direction there.

Let go of expectation on quick responses. We are a global economy run by volunteers. If you're lucky a company pays one person or a group of people to watch for contribution questions.

how to play well with others

- team dynamics discussion
- global dynamics discussion
- pair-programming discussion
- topics from the group discussion

live demo + practice

Link to this talk
https://github.com/stacybird/talks/blob/master/20190910*git*201*how*to*play*well*with*others.md

Let us work through some of these together!

Thank you!

Reference links:

https://git-scm.com/docs (complete git command reference)

https://git-scm.com/book/en/v2 (the book by chapter)

https://git-scm.com/docs/git-rebase (more detail on rebase)