Sergio Pichardo's Git Presentation (2.2 — 2.4)

Friday, November 12, 2021

TOPICS

- short status (`git status -s`)
- Removing files (`git rm`)
- Moving files (`git mv`)
- Viewing commit history (`git log –pretty=format`)
- Limiting output (`git log –since=2.weeks`)

Git Short Status

Example 1: Untracked files status (??)

```
# EXAMPLE SETUP
ls
notes.md      portfolio    untracked-portfolio


# create the project files
mkdir 01_files_not_tracked
cd 01_files_not_tracked

# initialize a git directory
git init .

# create alias
alias gss="git status -s"

# show state of git project
?? .editorconfig
?? .gitignore
?? LICENSE
?? README.md
?? dist/
?? package-lock.json
?? package.json
?? scripts/
?? src/

# add to the staging area
git add README.md


# show untracked files
git status -s

A .editorconfig
```

```
?? .gitignore
?? LICENSE
?? README.md
?? dist/
?? package-lock.json
?? package.json
?? scripts/
?? src/

# start tracking files
git add *

# show tracked files
git status -s

# show staging area aka .git/index file
cat .git/index

~ɱ@_P����⛲��CK�)�wZ���S�
�@_R����⛲��CK�)�wZ���S�
�cripts.�@_Q����⛲��CK�)�wZ���S�
styles.css.c�n�Dd�M���A�M}�
```

Example 2: Show files modified while in the staging area

```
# EXAMPLE SETUP

# create the project files
cp -r portfolio-untracked/ 02_file_modified_while_in_staging_area--AM/
cd 02_file_modified_while_in_staging_area--AM

# initialize a git directory
git init .

# ------------
# show status
git status -s

??  README.md
?? .editorconfig
?? .gitignore
?? LICENSE
?? dist/
?? package-lock.json
?? package.json
?? scripts/
?? src/

# add README to staging area
git status -s

A  README.md
?? .editorconfig
?? .gitignore
?? LICENSE
?? dist/
?? package-lock.json
?? package.json
?? scripts/
?? src/
```

```
# modify file
echo "hello world" >> README.md

# show state of files
git status -s

AM  README.md
?? .editorconfig
?? .gitignore
?? LICENSE
?? dist/
?? package-lock.json
?? package.json
?? scripts/
?? src/
```

Example 3: Show staged and Ignored Files

```
# EXAMPLE SETUP
# create the project files
cp -r untracked-portfolio 03_ignored_files && cd $_

# initialize a git directory
git init .

# show current files
git status -s


# add file to .gitingore
vim .gitignore
# -- add '.editorconfig' to the .gitignore file

# show status
git status -s

?? .gitignore
?? LICENSE
?? README.md
?? dist/
?? package-lock.json
?? package.json
?? scripts/
?? src/
!! .editorconfig
```

Example 4: A staged file was deleted from the working tree

```
# EXAMPLE SETUP
# create the project files
mkdir 04_file_deleted_while_in_index && cd $_

# initialize a git directory
git init .
```

```
# create a file
echo 'hello world' >> temp.txt

# show state
git status -s

?? temp.txt

# add temp to staging area
git add temp.txt

# show current state of git project
git status -s

A  temp.txt

# remove file while in staging area
trash temp.txt

# show status
AD temp.txt
```

Example 5: A previously committed file was modified, then staged, then it was modified in the staging area.

```
# EXAMPLE SETUP
# create the project files
mkdir 05_committed_file_modified_in_index && cd $_

# initialize git project
git init

# create a test file
echo 'hello world' >> temp.txt

# show current project status
git status -s

?? temp.txt

# add temp.txt to staging area
git add .

# show current state
git status -s

A  temp.txt

## commit the temp.txt
git commit -m "add temp.txt file"


## modify the temp.txt file
echo 'hello world again!' >> temp.txt

## show status
''M temp.txt

# stage file
git add .
```

```
# show status
git status -s
M'' temp.txt


# modify the file
echo 'hello mundo!' >> temp.txt
git status -s

## show status
MM temp.txt
```

CHALLENGE

Example 6: File was committed, then it was renamed (MD)

Renamed in staging area and deleted in working-tree

```
# EXAMPLE SETUP
# create the project files
mkdir 06_committed_file_renamed_in_index && cd $_

# initialize a git directory
git init .

# create a file
echo 'hello world' >> temp.txt

## Try to recreate the MD status
# -----------------------
```

Tell people to experiment with different combinations and show them the man pages for git-status

```
man git-status

# press PageDn or PageUp to move through man page
```

Remove files from Staging Area

- To remove a file from Git, you have to remove it from the staging area and then commit.
- The `git rm` command removes files from your staging area and also removes files from your working directory so you don't see it as an untracked file the next time around.
- If you simply remove the file from the working directory, it shows up under the "Changes not staged for commit" (unstaged) of your git status output:

```
git init .
touch index.html
git add index.html # stage the file

rm index.html
git status -s

AD index.html

git status
# it will show the status in two places
# Changes to be committed
  new file: index.js
# Changes not staged for commit:
  deleted: index.js

git add index.html
# this is not really what we want
# we can add that change to the staging area
# what git will do is to simply discard the file
# from both the staging area and the working tree
```

To remove the file from the **staging area** and the **working directory**

```
# this command is the equivalent of the steps
# we took to remove the file in the previous example
git rm --force index.html
```

To remove files from the **staging area**, but **keep them** in the **working directory**

```
# removing all files with the .md file extension
touch {1..10}.js
git add ./*.js
git rm ./\*.md --cached
```

Moving files (Renaming files tracked by Git)

If you want to rename a staged file you do it with `git mv`

```
# file setup
git init .
touch read.me
git add read.me

# change the name of a file in the staging area
git mv read.me README.md
```

```
# explicitly print the rename action
git mv README.md README --verbose
```

`git mv` is a convenience command and the same can be achieved with:

```
# this is the equivalent of doing something like
mv README.md README
git rm README.md
git add README
```

Viewing Commit History

The `git log` command allows you to view your commit history

⭐Project Setup

```
# install the simplegit repo
git clone https://github.com/schacon/simplegit-progit
cd simplegit-progit


# clone open source portfolio git project
git clone https://github.com/StartBootstrap/startbootstrap-freelancer.git portfolio
cd portfolio
```

`git log`

- By default, when you run `git log` with no arguments, it'll list the commit history in reverse chronological order (most recent commits first).
- This command lists each commit with:
  - The commit's SHA-1 checksum
  - The Author's name
  - The Author's email
  - The date written
  - The commit message

```
git log

# output
commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD)
Author: Scott Chacon <schacon@gmail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
```

```
    changed the verison number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit
```

`git log --patch` or `git log -p`

- Shows you the difference introduced in each commit (aka the **patch**)

```
git log --patch

# output omitted
# the patch is denoted by the green line with the `+` symbol
```

`git log -p -1`

- You can limit the number of log entries to list with a number option
- This option displays the same information but with a diff directly following each entry.
- This is very helpful for code review or to quickly browse what happened during a series of commits that a collaborator has added.

```
git log --patch -1
```

⭐ `git log --stat`

- You can use `git log --stat` to see abbreviated stats for each commit.
- Prints below each commit entry a list of modified files, how many files were changed, and how many lines in those files were added or removed.
- It also puts a summary of information at the end

```
git log --stat

commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD)
```

```
Author: Scott Chacon <schacon@gmail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700

    changed the verison number

 Rakefile | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 16:40:33 2008 -0700

    removed unnecessary test code

 lib/simplegit.rb | 5 -----
 1 file changed, 5 deletions(-)

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700

    first commit

 README           |  6 ++++++
 Rakefile         | 23 +++++++++++++++++++++++
 lib/simplegit.rb | 25 +++++++++++++++++++++++++
 3 files changed, 54 insertions(+)
```

`git log --pretty=oneline`

- The `—-pretty` option changes the log output to formats other than the default.
- The `oneline` value for this option prints each commit on a single line, which is useful if you're looking at a lot of commits.

```
git log --pretty=oneline

ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD) changed the verison number
085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7 removed unnecessary test code
a11bef06a3f659402fe7563abf99ad00de2209e6 first commit


# similar to


git log --oneline

ca82a6d (HEAD -> master, origin/master, origin/HEAD) changed the verison number
085bb3b removed unnecessary test code
a11bef0 first commit


# `git log --oneline` is a shorthand for
# `git log --pretty=oneline --abbrev-commit`
```

`git log --pretty=short`

```
git log --pretty=short

commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD)
Author: Scott Chacon <schacon@gmail.com>

    changed the verison number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gmail.com>

    removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>

    first commit
```

`git log --pretty=full`

```
git log --pretty=full

commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD)
Author: Scott Chacon <schacon@gmail.com>
Commit: Scott Chacon <schacon@gmail.com>

    changed the verison number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
Author: Scott Chacon <schacon@gmail.com>
Commit: Scott Chacon <schacon@gmail.com>

    removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gmail.com>
Commit: Scott Chacon <schacon@gmail.com>

    first commit
```

`git log --pretty=fuller`

```
git log --pretty=fuller

commit ca82a6dff817ec66f44342007202690a93763949 (HEAD -> master, origin/master, origin/HEAD)
Author:     Scott Chacon <schacon@gmail.com>
AuthorDate: Mon Mar 17 21:52:11 2008 -0700
Commit:     Scott Chacon <schacon@gmail.com>
CommitDate: Fri Apr 17 21:56:31 2009 -0700

    changed the verison number

commit 085bb3bcb608e1e8451d4b2432f8ecbe6306e7e7
```

```
Author:     Scott Chacon <schacon@gmail.com>
AuthorDate: Sat Mar 15 16:40:33 2008 -0700
Commit:     Scott Chacon <schacon@gmail.com>
CommitDate: Fri Apr 17 21:55:53 2009 -0700

    removed unnecessary test code

commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author:     Scott Chacon <schacon@gmail.com>
AuthorDate: Sat Mar 15 10:31:28 2008 -0700
Commit:     Scott Chacon <schacon@gmail.com>
CommitDate: Sat Mar 15 10:31:28 2008 -0700

    first commit
```

`git log --pretty=format:"<specifiers go here>"`

- The `format` option value allows you to specify your own log output format based on special specifiers.

```
git log --pretty=format:"%h - %an, %ar : %s"

ca82a6d - Scott Chacon, 14 years ago : changed the verison number
085bb3b - Scott Chacon, 14 years ago : removed unnecessary test code
a11bef0 - Scott Chacon, 14 years ago : first commit
```

Specifiers for `git log --pretty=format`

(Specifier code | Description of Output)

`%H`: Commit hash

`%h`: Abbreviated commit hash

`%T`: Tree hash

`%t`: Abbreviated tree hash

`%P`: Parent hashes

`%p`: Abbreviated parent hashes

`%an`: Author name

`%ae`: Author email

`%ad`: Author date (format respects the —date=option)

`%ar`: Author date, relative

`%cn`: Committer name

`%ce`: Committer date

`%cr`: Committer date, relative

`%s`: Subject

What's the difference between **author** and **committer**?

- **Author**: The person who originally wrote the work
- **Committer**: The person who last applied the work
- So, if you send in a patch to a project and one of the core members applies the patch, both of you get credit — you as the author, and the core member as the committer.

⭐ `git log --pretty=format:"%h %s" --graph`

- the `--graph` option adds a little ASCII graph showing your branch and merge history:

```
git log --pretty=format:"%h %s" --graph

* 2d3acf9 Ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Add method for getting the current branch
* | 30e367c Timeout code and tests
* | 5a09431 Add timeout protection to grit
* | e1193f8 Support for heads with slashes in them
|/
* d6016bc Require time for xmlschema
*  11d191e Merge branch 'defunkt' into local
```

Common options for git log

Option | Description

`-p`: Show the path introduced with each commit

`--stat`: Show statistics for files modified in each commit

`--shortstat`: Display only the changed/insertions/deletions line from the —stat command

`--name-only`: Show the list of files modified after the commit information.

`--name-status`: Show the list of files affected with added/modified/deleted information as well.

`--abbrev-commit`: Show only the first few characters of the SHA-1 checksum instead of all 40.

`--relative-date`: Display the date in a relative format (for example, “2 weeks ago”) instead of using the full date format.

`--graph`: Display an ASCII graph of the branch and merge history beside the log output.

`--pretty`: Show commits in an alternate format. Option values include oneline, short, full, fuller, and format (where you specify your own format).

`--oneline`: Shorthand for `-pretty=oneline --abbrev-commit` used together.

Limiting Log Output

⭐ `git log --since=2.weeks`

- Get a list of commits made in the last two week

```
git log --since=2.weeks
```

`git log --until=<YYYY-MM-DD>`

- Shows you commits that were created on a specific date

```
git log --until=2021-10-31
```

⭐ `git log --author="Sergio"`

- Log commits based on author

```
git log --author="Sergio"
```

⭐ `git log --grep="README"`

- Filtering commits based on keyword in the commit

```
git log --grep="README"
```

`git log -- path/to/file`

- Filtering commits by passing a path
- If you specify a directory or file name, you can limit the log output to commits that introduced a change to those files. This is always the last option and is generally preceded by double dashes (`-`) to separate the paths from the options
- Resources →

```
git log -- path/to/file
```

Options to limit the output of `git log`

(Options | Description)

`-<n>`: Show only the last n commits

`--since`, `--after`: Limit the commits to those made after the specified date.

`--until`, `--before`: Limit the commits to those made before the specified date.

`--author`: Only show commits in which the author entry matches the specified string.

`--committer`: Only show commits in which the committer entry matches the specified string.

`--grep`: Only show commits with a commit message containing the string

`-S`: Only show commits adding or removing code matching the string

`git log --pretty="%h - %s" --author='Junio C Hamano' --since="2008-10-01" \ --before="2008-11-01" --no-merges -- t/`

```
git log --pretty="%h - %s" \
--author='Junio C Hamano'
--since="2008-10-01" \
--before="2008-11-01" \
--no-merges -- t/

5610e3b - Fix testcase failure when extended attributes are in use
acd3b9e - Enhance hold_lock_file_for_{update,append}() API
f563754 - demonstrate breakage of detached checkout with symbolic link HEAD
d1a43f2 - reset --hard/read-tree --reset -u: remove unmerged new paths
51a94af - Fix "checkout --track -b newbranch" on detached HEAD
b0ad11e - pull: allow "git pull origin $something:$current_branch" into an unborn branch
```