Try-Git Documentation

Release 0.2

sshquack

July 20, 2014

Contents

A fast walk-through on the Git distributed version control system (DVCS). If you have used `hg` in the past, then you may find that `git` and `hg` share a lot of common ideas. So you may notice the Mercurial equivalent commands for easier understanding.

Live docs	http://try-git.readthedocs.org/en/latest/index.html
View source	https://github.com/sshquack/try-git
Content license	Public domain
RTD theme license	MIT (https://github.com/snide/sphinx_rtd_theme)

Note: This document is written in `reStructured text` markup - same as the Python docstrings. So feel free to clone, edit, and generate PDF using Sphinx.

CHAPTER 1

Contents

1.1 Bootstrap

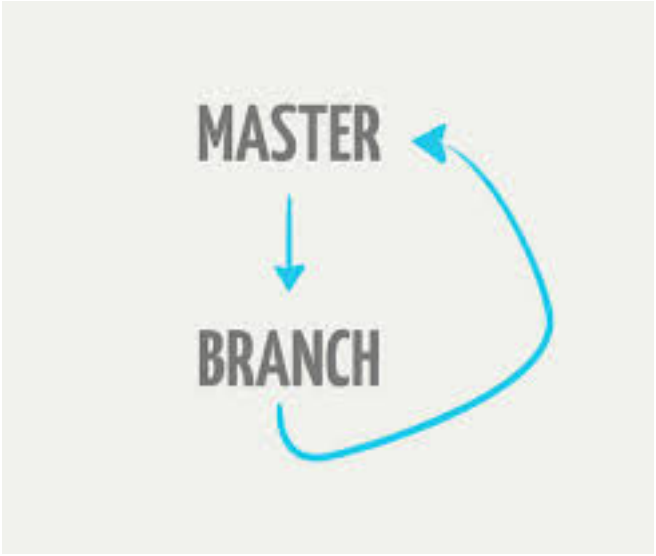
1. List all git commands `git help`
2. Get specific command man page `git help <command>`
3. Initialize repo `git init [directory]`
4. Check status `git status`
   (a) For mercurial like short concise status use `git status -sb`
5. Add file `git add <filename>`
6. Add files `git add <dirname>/<*.ext>`
   (a) Directory level wildcard globs are supported
7. Remove files `git rm <*.ext>`
   (a) `rm` -> `remove` in Mercurial
8. Commit `git commit -m "<message>"`
9. View log `git log`
   (a) `git log --graph` -> `hg glog`
   (b) `git log --graph --source` will show per-commit branch info on `master`
   (c) `git log --graph --source --all` shows commits across all branches
   (d) `git log --graph --source <branch-name>` shows commits on specific branch
10. Add a remote origin repo `git remote add origin https://github.com/sshquack/try-git.git`
    (a) `origin` is the pointer (or name) to the remote repo
    (b) A cloned repo has a local branch like `master` and tracking branch `origin/master`
    (c) Each tracking branch will have 1:1 map to a local branch (unless local is deleted)
    (d) Developers can have as many local branches without other developers knowing about it. One can still make commits to local branches. But all commits to local branches will stay local and `git push` will not have a remote tracking branch to save.

    Note: If your local repo directory is wiped out, then the local branches would be wiped out as well.

11. List all tracked remote repository names with `git remote -v show` and information about a specific remote repository name with `git remote -v show origin`
12. Push to remote repo `git push -u origin master`
    (a) `-u` - Remembers parameters by setting upstream reference. Skip it on next `git push`
    (b) `master` -> `default` branch in Mercurial
    (c) `master` is the local branch and `origin/master` is the remote tracking branch
13. Pull remote changes `git pull origin master`
14. View diff against pulled changesets `git diff HEAD`
    (a) `HEAD` == `tip` in Mercurial
15. View diff against staged files `git diff --staged` or on unstaged yet modified files `git diff`
    (a) Added uncommitted files are called `staged` files
    (b) `git diff` == `hg diff`
16. Check the current commit hash `git rev-parse --short HEAD`
17. Reset staged changes :code:`git reset <dirname>/<filename>`
    (a) `git reset --hard HEAD` == `hg revert`
    (b) `git reset --hard <commit-hash>` == `hg rever -c <commit-hash>`

    Note: `reset` will not work on public repos. Use `checkout` + `commit` + `push` for this use case.

18. Get rid of all the changes since the last commit of a file :code:`git checkout -- <filename>`
    (a) `git checkout` == `hg update` updates to recent revision in `HEAD`
    (b) `git checkout .` == `hg update --clean` will discard changes on all unstaged files
    (c) `git clean -dfx` == `hg purge --all` will clean untracked files (and dirs)
    (d) `git checkout <commit-hash>` == `hg update <commit-hash>`
19. Pick `git-flow` or the simple branching model



20. View current or all branches. Switch to the newly created branch.

   (a) `git branch` == `hg branch` + `hg branches`

   (b) `git branch -a` == `hg branches`

21. Create a feature branch and switch to it.

   (a) `git branch <feature-branch>`

   (b) `git checkout <feature-branch>`

22. Pull, view, and update branch changes

   (a) `git fetch` == `hg pull`

   (b) `git fetch && git log HEAD..origin/<branch-name> --patch --source` == `hg incoming --patch`

   (c) `git pull` == `hg pull && hg update`

23. Commit changes and push

   (a) `git commit -m <message>` == `hg commit -m <message>`

   (b) `git fetch && git log origin/<branch-name>..HEAD --patch --source` == `hg outgoing --patch`

   (c) `git push -u origin <feature-branch>` == `hg push` to publish remote branch and setup tracking

   (d) `git log -n 1 --source` to verify changeset

   (e) `git show -s` == `hg tip` to verify commit message

24. Merge branches

   (a) Switch to release branch `git checkout <release-branch>` or master :code‘git checkout master‘

   (b) Merge and resolve conflicts `git merge <feature-branch>`

   (c) Delete (or cleanup) a local branch `git branch -d <feature-branch>` and from remote too `git push origin :feature-branch`

25. Tag the commit, push upstream, and sync up with remote tags

   (a) `git tag v0.1`

   (b) `git push --tags`

   (c) `git fetch --tags --prune`

26. GUI tools. When something doesn't see right, use the GUI. They are a great learning tool.

   (a) `gitk` or `gitg` to view current repo state

   (b) From inside `gitk` you can open `git-gui` in Files -> Start git gui. This allows you to stage files, commit, add, remove and a ton of other things.

   (c) In the Mercurial world `thg` is the defacto standard

1.1.1 Git <-> Mercurial cheatsheet

http://mercurial.selenic.com/wiki/GitConcepts#Command_equivalence_table

1.2 Typical Workflow

Initialize a Git repository locally and later one can choose to add the local repository as a part of the remote repository (think GitHub).

Before proceeding create a new repo at https://github.com/repositories/new

```
# Create a directory for a project called "Hello-World" in the user directory
mkdir ~/Hello-World

# Change the current working directory to the newly created directory
cd ~/Hello-World

# Set up the necessary Git files
# Initialized empty Git repository in /Users/you/Hello-World/.git/
git init

# Create a file called "README" in your Hello-World directory
touch README.rst

# Stage your README file, adding it to the list of files to be committed
git add README.rst

# Commit your files, adding the message "first commit"
git commit -m 'first commit'

# Create a remote named "origin" pointing at your GitHub repository
git remote add origin https://github.com/username/Hello-World.git

# Send your commits in the "master" branch to GitHub
git push -u origin master
```

Refer: https://help.github.com/articles/create-a-repo#create-a-readme-for-your-repository
http://stackoverflow.com/questions/5561295/what-does-git-push-u-mean

GitHub also maintains a set of .gitignore (like .hgignore) files depending on the programming language used for that repo. https://github.com/github/gitignore - Be sure to check what is being ignored.

1.3 FAQs

- Question: **Where can I find the `~/.hgrc` like config?**

  Answer: http://git-scm.com/book/en/Customizing-Git-Git-Configuration

  - System-wide setting are in `/etc/gitconfig`
  - User-wide settings are in `~/.gitconfig`
  - All changes made using the `git config --global` will show up in the `~/.gitconfig` file.

- Question: **Can I enable bash colors for Git console output like Mercurial?**

  Answer: http://git-scm.com/book/en/Customizing-Git-Git-Configuration#Colors-in-Git

  - `git config --global color.ui true`
  - `git config --global --get color.ui`
  - `git config --global -list` to view all global configs

  - `git config --local -list` to view all repo local configs

- Question: **Is there a difference between** `git pull` **and** `git fetch` **?**

  Answer: http://stackoverflow.com/questions/292357/whats-the-difference-between-git-pull-and-git-fetch

  - `git pull` = `git fetch` + `git merge`
  - `git pull` = `hg pull` + `hg update`
  - `git fetch` = `hg pull`
  - `git fetch` gathers any commits from the target branch that do not exist in your current branch and stores them in your local repository.
  - `git pull` first fetches incoming commits and automatically merges the commits without letting you review them first. Alternatively, one can make `git pull` do a `git fetch` + `git rebase` as well.

- Question: **What is the equivalent of** `hg incoming` **and** `hg outgoing` **?**

  Answer: http://stackoverflow.com/questions/1331385/how-can-i-see-incoming-commits-in-git

  - `git fetch` to gather incoming commits
  - `git fetch && git log ..origin/<branch-name>` == `hg incoming`
  - `git fetch && git log origin/<branch-name>..` == `hg outgoing`
  - `git diff --stat ..origin/<branch-name>` to view incoming text diff statistics == `hg incoming --stat`
  - `git diff --stat origin/<branch-name>..` to view incoming text diff statistics == `hg outgoing --stat`
  - `git whatchanged ..origin/<branch-name> -p` to view incoming text patch == `hg incoming --patch`
  - `git whatchanged ..origin/<branch-name> -p` to view incoming text patch == `hg outgoing --patch`

- Question: **How to revert to a previous commit or changeset?**

  Answer:

  - Switch to the correct branch with `git checkout <branch-name>`
  - Revert to old commit with `git revert --hard <SHA-hash>`

- Question: **How to create a remote branch from local branch while pushing?**

  Answer: http://stackoverflow.com/questions/1519006/git-how-to-create-remote-branch

  - `git push -u origin <branch-name>`

- Question: **How to delete a branch even if it was not merged?**

  Answer: http://stackoverflow.com/questions/2003505/how-do-i-delete-a-git-branch-both-locally-and-remote

  - `git branch -D <branch-name>`

- Question: **Explain** `git rebase` **for humans.**

  Answer: http://stackoverflow.com/questions/2672351/hg-how-to-do-a-rebase-like-gits-rebase

```
1. Start working on new feature:
$ git co -b newfeature-123  # (a local feature development branch)
do a few commits (M, N, O)
```

```
master A---B---C
                \
newfeature-123   M---N---O

2. Pull new changes from upstream master:
$ git pull
(master updated with ff-commits)

master A---B---C---D---E---F
                \
newfeature-123   M---N---O

3. Rebase off master so that my new feature
can be developed against the latest upstream changes:
(from newfeature-123)
$ git rebase master

master A---B---C---D---E---F
                            \
newfeature-123               M---N---O
```

CHAPTER 2

Resources

Most of the information in this doc have been aggregated from these following excellent resources. Please refer to their appropriate licenses.

Beginner:

- 15 minute walkthrough - http://try.github.io/levels/1/challenges/1
- Learm Git - http://www.git-tower.com/learn/
- Know the basics - https://www.atlassian.com/git/tutorial/git-basics
- Quick cheatsheet - https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
- Learn branching in Browser - http://pcottle.github.io/learnGitBranching/?demo
- How Github uses Git #NSFW - https://www.youtube.com/watch?v=qyz3jkOBbQY

Intermediate:

- Git Pro - http://git-scm.com/book
- Simple Branching - https://gist.github.com/jbenet/ee6c9ac48068889b0912
- Exhaustive cheatsheet - http://gitready.com/

Topic - Workflows:

- git-flow 1 - http://nvie.com/posts/a-successful-git-branching-model/
- git-flow 2 - http://jeffkreeftmeijer.com/2010/why-arent-you-using-git-flow/
- https://www.atlassian.com/git/workflows
- http://gitolite.com/gcs.html

Topic - Clients:

- Git Tower (Mac) - http://www.git-tower.com
- GitHub (Mac) - http://mac.github.com/
- Gitk / Git-gui - http://git-scm.com/docs/gitk | http://git-scm.com/docs/git-gui
- Git-Cola (Linux) - https://github.com/git-cola/git-cola
- Ungit (node.js) - https://github.com/FredrikNoren/ungit
- Git Legit (CLI) - https://github.com/kennethreitz/legit

CHAPTER 3

Additional Information

If you can't find the information you're looking for, have a look at the index or try to find it using the search function:

- *genindex*
- *search*