*Learn Git in a Month of Lunches*

Commands

List all the commands

$ git [switches] <command> [<args>]

List the files in the repository

$ git ls-files

$ git --version

$ git config -–system --list

$ git config -–global --list

$ git config -–system --list

Add

$ git add -–dry-run . //Showing what it would done//

$ git add . //

$ git add –p

Reset

To undo a staging area change, you have to use git reset, which is the opposite of git add.

$ git reset file01.txt //To undo the staging area change that you made

Resetting a file to the last committed version

Check out the latest committed version of the file into your working directory

$ git checkout -- math.sh

Pagination

$ git –p log

Diff

Show any changes between the tracked files in the current directory and the repository

$ git diff

Show any changes between the staging area and the repository

$ git diff --staged

Log

$ git log --stat

$ git log --shortstat

$ git log --oneline

$ git log --parents –-abbrev-commit

$ git log --patch

$ git log --patch-with-stat

$ git log --author=luc

$ git rev-parse HEAD

$ git rev-parse master

Tag

$ git tag name –m message SHAID

Delete the master branch, using the -d switch to git branch:

$ git branch -d master



The starting point that you used in the TRY IT NOW section was the branch you created before. There’s nothing wrong with using a current branch as a starting point! The starting point can also be a commit SHA1 ID or a tag.

List all branches.

git branch

Create a new branch named dev. (This branch points to the same commit as HEAD.)

git branch dev

Change your working directory to the branch named dev.

git checkout dev

Delete the branch named master.

git branch -d master

View history of the repository across all branches (see section 9.2.3).

git log --graph –decorate --pretty=oneline --all --abbrev-commit

Make an alias named lol for the git log command in the previous row (see the Above

and Beyond sidebar).

git config --global alias.lol "log --graph --decorate --pretty=oneline --all --abbrev-commit"

List all branches with SHA1 ID information.

git branch -v

Make a branch using YOUR\_SHA1ID as the starting point.

git branch fixing\_readme YOUR\_SHA1ID

Make a branch named another\_fix\_branch using branch fixing\_readme as the starting point.

git checkout -b another\_fix\_branch fixing\_readme

Show a record of all the times you changed branches (via git checkout).

git reflog

Set the current work in progress (WIP) to a stash (holding area), so you can perform a git checkout.

git stash

git stash list List works in progress that you’ve stashed away.

git stash pop Apply the most recently saved stash to the current working directory; remove it from the stash.

***Checking the difference between two branches***

The git diff command has an interesting syntax you can use to determine the differences between two branches.

git diff master...bugfix

$ git config --global merge.tool Kdiff3

Indicate the difference between BRANCH1 and BRANCH2 relative to when they first became different.

git diff BRANCH1...BRANCH2

Summarize the difference between BRANCH1 and BRANCH2, by listing each file and its status.

git diff --name-status BRANCH1...BRANCH2

Abandon a merge between two conflicted branches.

git merge –abort

Show the base commit between BRANCH1 and BRANCH2.

git merge-base BRANCH1 BRANCH2

git branch --all Show remote-tracking branches in addition to local branches.

Viewing branches in your clone

Knowing the branches available in a repository is important. Here’s the command-

line equivalent to get a simplified list of branches

git log --simplify-by-decoration --decorate --all –oneline

The Git command git ls-tree lists all the files in a tree. The git ls-tree command is a helpful way to get the list of files for any part of your Git history.

git ls-tree –r HEAD

*Collaborating with remotes*

The simplest form of the git remote command shows only the name of the remote

$ git remote

$ git remote -v

Now use git remote to rename *origin*:

git remote rename origin beginning

Adding a remote

Add a remote named bob that points to the local repositoryin ../math.bob.

git remote add bob ../math.bob

Interrogating (querying) the remote repository by exercising the git ls-remote command. This command returns a list of the SHA1 IDs of each branch and tag (each reference) on the remote repository.

You can compare this to SHA1 IDs of the current local repo (math.carol) by typing the following:

git ls-remote .

In this command, the period (.) represents the current local repository. This should give you the same listing!

*Pushing your changes*

Making comparisons in this fashion is a little tedious (git ls-remote) . A better way is to use the git remote command’s show subcommand

git remote –v show origin

pushing a new branch

The command will not only push the branch, but also create the remote-tracking branch in your local repository.

git push --set-upstream origin new\_branch.

You need to run git push with the --set-upstream switch only once. When you use this switch, Git writes this information into a configuration file.

git config --get-regexp branch

This command retrieves configuration settings that match the string branch. The meaning of --get-regexp is *get regular expression*, which means git config will match any configuration with the word *branch* in it.

***Deleting branches on the remote***

git push origin :new\_branch

Using a colon causes the git push command to use the more complex meaning of the fourth argument



the fourth argument to git push is a *refspec*, a colon-separated string that describes the mapping between a source and destination branch

Using git push origin master is equivalent to using git push origin master:master

When you omit src from the full form, leaving the string :master, you’re telling Git to delete that remote branch

***Pushing and deleting tags***

You’re going to tag the commit that is two commits behind the HEAD (which is the last commit). The arcane syntax (HEAD^^) says from HEAD, go two commits back

git tag -a two\_back -m "Two behind the HEAD" HEAD^^

To push your tag, type this:

git push origin two\_back

Delete this tag, and confirm that it’s no longer on the remote:

git push origin :two\_back

Your local tag still exists. To delete this, use git tag with the -d switch:

git tag -d two\_back

Using git pull: a two-part operation

git pull is a two-part operation. In its default mode, git pull is shorthand for git fetch followed by git merge FETCH\_HEAD

The second step of git pull is to run git merge FETCH\_HEAD. What is FETCH\_HEAD? It’s a reference to the

remote branch that you just fetched in the previous section



git rev-parse FETCH\_HEAD

This should give you a SHA1 ID that points to the latest commit from the remote’s master branch. Keep in mind that this remote master branch is already in your local repository. You can access it by its special name, origin/

master.

git rev-parse origin/master

This last command should give you the same SHA1 ID as FETCH\_HEAD.

you can ask Git to indicate what is different between these two branches,

git diff HEAD..FETCH\_HEAD

Finally, and most important, FETCH\_HEAD is used as the argument to git merge

git merge FETCH\_HEAD

Restricting pulls to fast-forwards only

You can run git pull to incorporate only commits that are descendants of your current branch, using the --

ff-only switch. This prevents Git from doing any automated work, unless it’s a fastforward merge.

git pull --ff-only

Software archaeology

Let’s identify from the git log output those commits that represent merges of two branches.

git log --parents –oneline

Rather than finding these merges by hand, you can use the --merges switch:

git log --merges

***Limiting the display of commits***

LIMITING BY FILE

you’ll limit the git log output to the commits that affected one or more files:

git log --oneline readme.txt

git log --oneline readme.txt renamed\_file

FINDING SPECIFIC COMMITS

You can tell git log to show only those commit messages that match a particular string by using the --grep switch to search commit messages for that string

git log --grep=change

This should list any commits that have the word *change* in the commit message

LIMITING BY TIME RANGE

git log --since 10/10/2014 --until 10/24/2014

LIMITING BY AUTHOR

You can also limit commits to a particular author, with the --author switch.

An easy way to get the list of authors from a repository is to use git shortlog.

After you have an author’s name or email address, you can restrict commits to that one author by passing it into the git log --author command.

git log --author="Rick Umali"

git log --author="Rick"

***Seeing differences with git log***

Use this command to display the number of files that have changed between the most current commit (HEAD) and its immediate predecessor (HEAD^):

git log --stat HEAD^..HEAD

The interesting syntax is the double-period. It specifies a revision range.

If you wanted to see what changed in the files since the last commit, try this:

git log --patch HEAD^..HEAD

It’s possible to combine patch and stat with --patch-with-stat

You can limit the output by typing the following:

git log --patch ef47d3f^..ef47d3f – file1

git log --patch ef47d3f^..ef47d3f – file1 file2

In these commands, you have to separate the files with a double-dash. When you type the commands, you’ll see that the git log output is limited to the files you specified.

This shows all the branches

git log --graph --decorate --oneline --all

git log --graph --decorate --oneline branch\_03

This command limits the git log output to one branch, which is easier to see.

git log --graph --decorate --oneline branch\_03 branch\_10 master

This form passes in three branches to the git log command

git name-rev 80f5738

The git name-rev command produces a name for a commit, based on the nearest branch.

Another way to determine what branch a commit belongs to is to use git branch --contains:

git branch -r --contains ce051a3

This command outputs the string origin/new\_feature, meaning that this SHA1 ID is part of the remote-tracking branch new\_feature. If you check out the new\_feature branch and type git branch --contains ce051a3, you’ll see the output new\_feature.

***Finding files of interest***

To limit your history to those commits that contained a specific string. You did this by using the git log --grep command. But this searches only the text in the commit messages. If you’re looking for files that contain a particular string, you’ll need to use the git grep command

git grep change

***Finding which revision updated a specific line of code***

git gui blame math.sh

git blame math.sh

*Understanding git rebase*

The .. syntax shows the commits between master and new\_feature.

git log --oneline master..new\_feature

***Using git reflog and git reset to revert your repo***

git reflog

To reset your repository back to the earlier state, type this:

git reset --hard HEAD@{4}

cleaning up history

git rebase --interactive master