Git Cheatsheet

Table of Contents

1) Basic Commands

1a) "init"

Create new project (creates a .git directory that tracks the project)

```
git init <projectName>
```

1b) "add"

Stage file for next commit...

```
git add <filename>
```

1c) "rm" (remove)

Tell git to stop tracking a file... (the file will not be deleted)

```
git rm <filename>
```

1d) "commit"

Commit staged changes...

```
git commit [-m <message>|]
```

Stage and commit all changes to tracked files (deleted and modified files, not new files)...

```
git commit -a [-m <message>]
```

1e) "status"

See all changes you've made since your last commit, and whether they are staged...

```
git status
```

1f) "show"

You can use git show:

```
$ git show REVISION:path/to/file
```

For example, to show the 4th last commit of the file src/main.c, use:

```
$ git show HEAD~4:src/main.c
```

Note that the path must start from the root of the repository. For more information, check out the man page for git-show.

(taken from http://stackoverflow.com/questions/338436/is-there-a-quick-git-command-to-see-an-old-version-of-a-file)

2) Branch Management

2a) "branch"

List available branches, with an asterisk next to the one you're working in...

```
git branch
```

Delete an existing branch...

```
git branch -d <branchname>
```

2b) "merge"

Merge all commits from another branch into this branch...

```
git merge <src-branchname>
```

Bring all commits from another branch into your current branch...

```
git merge <branchname>
```

2c) Merge Specific File or Commit

Git's facilities for merging specific files or commits from another branch are highly limited. Git wants to merge entire branches. However, in some instances, described below, it is possible to merge one or more files from another branch without too much effort. See the discussion in section 5 on Merging Between Branches if one of the cases described here is not appropriate.

The processes described below are error prone and tedious. It is highly recommended that you make a backup of the files before you attempt to merge them. After the merge be sure to inspect and make sure git did what you wanted.

Case 1: File(s) have changed in only one branch

If the files that you wish to merge have only changed in one of the two branches since the last time the files were in sync the files in the different branches can be brought into sync again with what is essentially a copy command. I.e. replacing the files contents from the current branch with the file's contents from another branch. To effect a copy from another branch use:

```
git checkout <branchname> -- <paths>
```

where *branchname* is the name of the branch from which you wish to copy, and *paths* are one or more files to copy.

Case 2: merging a single commit

If changes have been made to the files in both branches, and If the files in need of merging **differ by a single commit** you can use the git cherry-pick command. To merge **all of the files updated by a single commit** issue the following:

```
git cherry-pick -x <hash-of-commit>
```

where *hash-of-commit* is the 40 character unique hash tag associated with the commit. The cherry-pick will perform a genuine merge of the files - changes that do not result in a conflict will be automatically merged, and conflicts will be identified in the offending files with "====". Any conflicts will have to be manually resolved (with an editor). Other approaches discussed below do not perform true merges.

Note, to find out which files are associated with a particular commit use the command below. Intuitive, right?

```
git diff-tree --no-commit-id --name-only -r <hash-of-commit>
```

Case 3: Patching selected files

If the files in need of merger are not isolated to a single commit (i.e. you can't use the cherry-pick command) they can be interactively patched together. This is a last resort for merging two files and this method is little better than doing a diff between the files and editing by hand. The command

```
git checkout --patch <branchname> -- <filename>
```

will place you in an interactive utility that will step through each of the changes allowing you to select which changes to accept or deny. All changes will have to be reconciled, even those that do not result in a conflict.

An alternative to the above patch process is presented on stack overflow as the following sequence of commands:

```
git checkout <branchname> -- <paths>...
git reset <paths>...
git add -p <paths>...
git commit -m "'Merge' these changes"
```

It's not at all clear why the four command sequence is better than the single one above, but maybe i'm missing something...

2d) "checkout"

Undo any changes you've made to tracked files in the current branch since the last commit...

```
git stash

git stash drop stash@0
```

Undo any changes to a tracked file in the current branch since the last commit...

```
git checkout -- <filename>
```

Switch to another branch (throws away any uncommitted changes to tracked files in the current branch, and changes the working tree to look like the checked-out branch)...

```
git checkout <branchname>
```

Bring in all changes to a specific file from another branch (they will not come in as a commit, just modifications)...

```
git checkout <branchname> <filename>
```

this is one way to resolve merge issues. You can

```
git checkout origin/<branchname> <filename>
```

to use the version on the remote.

3) Remote Management

3a) "push"

Push any new commits to sourceforge...

```
git push origin <branch (normally master)>
```

3b) "pull"

Get any new commits from sourceforge...

```
git pull origin <branch (normally master)>
```

3c) "clone"

Get the repository for the first time...

```
git clone ssh://<sf-username>@git.code.sf.net/p/vapor/git <local-path-to-repo>
```

3c) "set-url"

Change the URL that your repository uses for the remote repository...

```
git remote set-url origin <new-url>
```

3d) "push a branch":

When you want to make a branch on the remote server, first create the branch in your local repository, for example using "git checkout -b branch_name". To push it to the remote server issue:

```
git push origin <branch_name>
```

3d) "pull a branch":

When you want to pull down a branch someone else created from the server, use the following command (generally the two branch names will be the same):

```
git pull origin <branch_name>:<desired_local_branch_name>
```

4) Analysis/Tracking

Three methods for seeing the difference (in commits) between local and remote...

```
git log origin..HEAD
git diff origin/master master
git log origin/master..master
```

View commit history and get commit hashes...

```
git log ...
```

-X will cause it to only log the last X commits
--pretty=oneline will make it easier to find a specific commit by message alone

log all commits from a to b (these would be SHA-1 hashes)...

```
git log <commit-a>..<commit-b>
```

show commits modifying a specific file

```
git log <filename>
```

Revert to the state of a given commit...

```
git reset --hard <commit-hash>
```

Suggested technique for observing previously committed code:

```
git branch branchToRevert
git checkout branchToRevert
git reset --hard <commit-hash>
git merge master // if the changes are suitable for merging with the master branch
```

Revert to the state of a specific 'tag'

```
git checkout <tag>
```

5) Advanced

5a) "stash" command - Sometimes, an incomplete task (task A) needs to be set aside for another task of higher priority (task B). The "stash" command allows you to switch your current working branch (to task B), without making a commit with incomplete code to task A.

Push the current state of the working directory onto a stack and return to a clean copy from the last commit...

```
git stash
```

Pop states off the stash stack...

```
git stash apply
```

Pop a state from somewhere else in the stack...

```
git stash apply@[depth] (where depth is the index in the stack of the item you want to pop)
```

View the stash stack...

```
git stash list
```

Remove items from the stash stack (without loading them)...

```
git stash drop stash@[depth]
```

5b) "tag" command - This is used to tag a certain commit with a name and/or annotations. It can be used to keep track of which commits represent a change in version number.

Tag the HEAD commit as X, with no annotations...

```
git tag X
```

View a list of existing tags in the repository...

```
git tag
```

Tag the HEAD commit as X, with annotations (note that the -m and message are optional. If these are not provided, git will open a text editor for you to enter the message)...

```
git tag -a X -m "Annotation Message"
```

But guess what? Tagging the code with 'git tag' isn't enough. To actually make the tag useful it must be pushed. Sigh. To actually use the tag in a useful way you must also execute:

```
git push --tags
```

Revert to the state of a specific 'tag'

```
git checkout <tag>
```

For additional details, visit this page: http://git-scm.com/book/en/Git-Basics-Tagging

5c) Selective merges between branches

Selectively merging between branches - merging a subset of files from one branch into another - is not directly supported by git. Git really only supports merging entire trees. With some effort a particular commit from one branch can be merged into another branch. There are two cases to consider. In either case BE SURE THE BRANCH YOU ARE MERGING INTO HAS BEEN COMMITTED BEFORE PROCEEDING!

1) A file *f* from branch A is to be merged into branch B, and the file f has only changed in branch A since A and B split. This is the simple case. Simply do:

```
git checkout B
git checkout B path_to_f
```

The above will overwrite, not merge, file f in branch B with the version of f from branch A

2) The far more complicated case is when file *f* has been edited in both branch A and B and now you want to merge the changes in branch B. This requires a multi-step process that is error prone and tedious, particularly if multiple files are involved. Moreover, if the file is part of a multi-file commit the entire commit will be merged (unless git voodoo is used to first split the single commit into multiple commits). To merge a random commit follow the instructions cut-n-paste below from stackoverflow merging selective files below. Note, the stackoverflow instructions don't mention that the identifiers such as 7266df7 are abbreviated (first 7 characters) from a git commit string. To map the abbreviated commit string to an actual file additional work is required with git-log.

You have some changes in branch 'feature' and you want to bring

some but not all of them over to 'master' in a not sloppy way (i.e.

you don't want to cherry pick and commit each one)

git checkout feature

git checkout -b temp

git rebase -i master

Above will drop you in an editor and pick the changes you want ala:

pick 7266df7 First change

pick 1b3f7df Another change

pick 5bbf56f Last change

Rebase b44c147..5bbf56f onto b44c147

#

Commands:

pick = use commit

edit = use commit, but stop for amending

squash = use commit, but meld into previous commit

#

If you remove a line here THAT COMMIT WILL BE LOST.

However, if you remove everything, the rebase will be aborted.

#

git checkout master

git pull . temp

git branch -d temp

So just wrap that in a shell script, change master into $to and change feature into $from and you are good to go:

#! /bin/bash

git-interactive-merge

from=$1

to=$2

git checkout $from

git checkout -b ${from}_tmp

git rebase -i $to

Above will drop you in an editor and pick the changes you want

git checkout $to

git pull . ${from}_tmp

git branch -d ${from}_tmp

6) Common Problems

6a) Remote not properly set - This can cause a few different kinds of symptoms. One is not being able to pull or push and having an error contacting the server. The other is an error where the server will "hang up unexpectedly" when attempting a push. To confirm that this is the problem, check 'git remote -v', which should tell you your fetch and push URLs for origin. If these URLs are not the same as the one visible in the Git tab on SourceForge, then you need to run the following command: 'git remote set-url origin <sourceforge-url>', which should repair the URLs, permitting you to push and pull again. Make sure you are logged in on SourceForge when you view the Git repository, so it can put your username in the URL.

6b) Throw away all local changes (committed and uncommitted) - The following procedure will reset all files in a given git repository tree to their state on the remote, throwing away ALL COMMITTED AND UNCOMMITTED LOCAL CHANGES to tracked files...

```
git fetch origin master

git reset --hard FETCH_HEAD
```

After performing this procedure, make sure to check for failures with 'git pull origin master', which should tell you that you are up-to-date. If not, you should probably re-clone for safety.