Git cheat sheets

Cheat sheets

- Alexander Zeitler's Git Cheat Sheet
- Zach Rusin's Git Cheat Sheet
- http://cheat.errtheblog.com/s/git/
- Nate Murray's Git Cheat Sheet
- Git – Der Spickzettel (in German)
- DZone Git RefCard Cheat Sheet

A practical git guide

Notes extracted from git screencast at http://www.peepcode.com.

Configuration

identify yourself to git: email and your name

```
git config --global user.name "David Beckwith"
```

```
git config --global user.email "dbitsolutions@gmail.com"
```

To view all options:

```
git config --list
```

OR

```
cat .git/config
```

Set up aliases

```
git config --global alias.co checkout
```

View your configuration

```
cat .gitconfig
```

To ignore whitespace (Ruby is whitespace insensitive)

```
git config --global apply.whitespace nowarn
```

Some nice aliases:

gb = git branch
gba = git branch -a
gc = git commit -v
gd = git diff | mate
gl = git pull
gp = git push
gst = git status

Start using git

```
git init
```

Ignoring files

Add a file in the root directory called `.gitignore` and add some files to it: (comments begin with hash)

*.log db/schema.rb db/schema.sql

Git automatically ignores empty directories. If you want to have a `log/` directory, but want to ignore all the files in it, add the following lines to the root `.gitignore`: (lines beginning with '!' are exceptions)

```
log/*
!.gitignore
```

Then add an empty `.gitignore` in the empty directory:

```
touch log/.gitignore
```

Scheduling the addition of all files to the next commit

```
git add .
```

Checking the status of your repository

```
git status
```

Committing files

```
git commit -m "First import"
```

Seeing what files have been committed

```
git ls-files
```

Scheduling deletion of a file

```
git rm [file name]
```

Committing all changes in a repository

```
git commit -a
```

Scheduling the addition of an individual file to the next commit

```
git add [file name]
```

Viewing the difference as you commit

```
git commit -v
```

Commit and type the message on the command line

```
git commit -m "This is the message describing the commit"
```

Commit and automatically get any other changes

```
git commit -a
```

A “normal” commit command

```
git commit -a -v
```

Viewing a log of your commits

```
git log
```

Viewing a log of your commits with a graph to show the changes

```
git log --stat
```

Viewing a log with pagination

```
git log -v
```

Visualizing git changes

```
gitk --all
```

Creating a new tag and pushing it to the remote branch

```
git tag "v1.3"
git push --tags
```

Creating a new branch

```
git branch [name of your new branch]
```

Pushing the branch to a remote repository

```
git push origin [new-remote]
```

Pulling a new branch from a remote repository

```
git fetch origin [remote-branch]:[new-local-branch]
```

Viewing branches

```
git branch
```

Viewing a list of all existing branches

```
git branch -a
```

Switching to another branch

The state of your file system will change after executing this command.

```
git checkout [name of the branch you want to switch to]
```

OR

```
git co [name of the branch you want to switch to]
```

Making sure changes on master appear in your branch

```
git rebase master
```

Merging a branch back into the master branch

First, switch back to the master branch:

```
git co master
```

Check to see what changes you’re about to merge together, compare the two branches:

```
git diff master xyz
```

If you’re in a branch that’s not the `xyz` branch and want to merge the `xyz` branch into it:

```
git merge xyz
```

Reverting changes to before said merge

```
git reset --hard ORIG_HEAD
```

Resolving conflicts

Remove the markings, add the file, then commit.

Creating a branch (and switching to the new branch) in one line

```
git checkout -b [name of new branch]
```

Creating a stash (like a clipboard) of changes to allow you to switch branches without committing

```
git stash save "Put a message here to remind you of what you're saving to the
clipboard"
```

Switching from the current branch to another

```
git co [branch you want to switch to]
```

Do whatever
Then switch back to the stashed branch

```
git co [the stashed branch]
```

Viewing a list of stashes

```
git stash list
```

Loading back the stash

```
git stash apply
```

Now you can continue to work where you were previously.

Deleting a branch (that has been merged back at some point)

```
git branch -d [name of branch you want to delete]
```

Deleting an unmerged branch

```
git branch -D [name of branch you want to delete]
```

Deleting a stash

```
git stash clear
```

Setting up a repository for use on a remote server

Copy up your repository. e.g.:

```
scp -r my_project deploy@yourbox.com:my_project
```

Move your files on the remote server to `/var/git/my_project`
For security make the owner of this project git
On the repository server:

```
sudo chown -R git:git my_project
```

Then (for security) restrict the “deploy” user to doing git-related things in `/etc/passwd` with a `git-shell`.

Checking out a git repository from a remote to your local storage

```
git clone git@yourbox.com:/var/git/my_project
```

Viewing extra info about a remote repository

```
cat .git/config
```

By virtue of having cloned the remote repository, your local repository becomes the slave and will track and synchronize with the remote master branch.

Updating a local branch from the remote server

```
git pull
```

Downloading a copy of an entire repository (e.g. `laptop`) without merging into your local branch

```
git fetch laptop
```

Merging two local branches (ie. your local xyz branch with your local master branch) USE MERGE

```
git merge laptop/xyz
```

This merged the (already copied laptop repository's xyz branch) with the current branch you're sitting in.

Viewing metadata about a remote repository

```
git remote show laptop
```

Pushing a committed local change from one local branch to another remote branch

```
git push laptop xyz
```

Creating a tracking branch (i.e. to link a local branch to a remote branch)

```
git branch --track local_branch remote_branch
```

You do not need to specify the local branch if you are already sitting in it.

```
git pull
```

Note: You can track(link) different local branches to different remote machines. For example, you can track your friend's "upgrade" branch with your "bobs_upgrade" branch, and simultaneously you can track the origin's "master" branch (of your main webserver) with your local "master" branch.

By convention, 'origin' is the local name given to the remote centralized server which is the way SVN is usually set up on a remote server.

Seeing which local branches are tracking a remote branch

```
git remote show origin
```

Working with a remote Subversion repository (but with git locally)

```
git-svn clone [http location of an svn repository]
```

Now you can work with the checked out directory as though it was a git repository. (cuz it is)

Pushing (committing) changes to a remote Subversion repository

```
git-svn dcommit
```

Updating a local git repository from a remote Subversion repository

```
git-svn rebase
```

NOTE: make sure you have your perl bindings to your local svn installation.

I screwed up, how do I reset my checkout?

```
git checkout -f
```

See also

- Git for computer scientists.
- Git user’s manual
- Git Magic
- Git Cheat Sheets Collection on DevCheatSheet.com