Git Cheat Sheet with git-flow feature

Git cheat sheet saves you from learning all the commands by heart.

Be free to contribute, update the grammar mistakes. You are also free to add your language file.

Git Cheat Sheet

Index

Setup

Show current configuration:

```
$ git config --list
```

Show local configuration:

```
$ git config --local --list
```

Show global configuration:

```
$ git config --global --list
```

Show system configuration:

```
$ git config --system --list
```

Set a name that is identifiable for credit when review version history:

```
$ git config --global user.name "[firstname lastname]"
```

Set an email address that will be associated with each history marker:

```
$ git config --global user.email "[valid-email]"
```

Set automatic command line coloring for Git for easy reviewing:

```
$ git config --global color.ui auto
```

Set global editor for commit

```
$ git config --global core.editor vi
```

Configuration Files

Repository specific configuration file [--local]:

```
<repo>/.git/config
```

User-specific configuration file [--global]:

```
~/.gitconfig
```

System-wide configuration file [--system]:

```
/etc/gitconfig
```

Create

Clone an existing repository:

There are two ways:

Via SSH

```
$ git clone ssh://user@domain.com/repo.git
```

Via HTTP

```
$ git clone http://domain.com/user/repo.git
```

Create a new local repository:

```
$ git init
```

Local Changes

Changes in working directory:

```
$ git status
```

Changes to tracked files:

```
$ git diff
```

Add all current changes to the next commit:

```
$ git add
```

Add some changes in <file> to the next commit:

```
$ git add -p <file>
```

Commit all local changes in tracked files:

```
$ git commit -a
```

Commit previously staged changes:

```
$ git commit
```

Commit with message:

```
$ git commit -m 'message here'
```

Commit skipping the staging area and adding message:

```
$ git commit -am 'message here'
```

Commit to some previous date:

```
git commit --date="`date --date='n day ago'`" -am "Commit Message"
```

Change last commit:

Don't amend published commits!

```
$ git commit -a --amend
```

Change commit date of last commit:

```
GIT_COMMITTER_DATE="date" git commit --amend
```

Change Author date of last commit:

```
git commit --amend --date="date"
```

Move uncommitted changes from current branch to some other branch:

```
git stash
git checkout branch2
git stash pop
```

Restore stashed changes back to current branch:

```
git stash apply
```

Remove the last set of stashed changes:

```
git stash drop
```

Search

A text search on all files in the directory:

```
$ git grep "Hello"
```

In any version of a text search:

```
$ git grep "Hello" v2.5
```

Commit History

Show all commits, starting with newest (it'll show the hash, author information, date of commit and title of the commit):

```
$ git log
```

Show all the commits(it'll show just the commit hash and the commit message):

```
$ git log --oneline
```

Show all commits of a specific user:

```
$ git log --author="username"
```

Show changes over time for a specific file:

```
$ git log -p <file>
```

Display commits that are present only in remote/branch in right side

```
$ git log --oneline <origin/master>..<remote/master> --left-right
```

Who changed, what and when in <file>:

```
$ git blame <file>
```

Branches & Tags

List all local branches:

```
$ git branch
```

List all remote branches:

```
$ git branch -r
```

Switch HEAD branch:

```
$ git checkout <branch>
```

Create and switch new branch:

```
$ git checkout -b <branch>
```

Create a new branch based on your current HEAD:

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch:

```
$ git branch --track <new-branch> <remote-branch>
```

Delete a local branch:

```
$ git branch -d <branch>
```

Force delete a local branch:

You will lose unmerged changes!

```
$ git branch -D <branch>
```

Mark the current commit with a tag:

```
$ git tag <tag-name>
```

Mark the current commit with a tag that includes a message:

```
$ git tag -a <tag-name>
```

Update & Publish

List all current configured remotes:

```
$ git remote -v
```

Show information about a remote:

```
$ git remote show <remote>
```

Add new remote repository, named <remote>:

```
$ git remote add <remote> <url>
```

Download all changes from <remote>, but don't integrate into HEAD:

```
$ git fetch <remote>
```

Download changes and directly merge/integrate into HEAD:

```
$ git remote pull <remote> <url>
```

Get all changes from HEAD to local repository:

```
$ git pull origin master
```

Get all changes from HEAD to local repository without a merge:

```
git pull --rebase <remote> <branch>
```

Publish local changes on a remote:

```
$ git push remote <remote> <branch>
```

Delete a branch on the remote:

```
$ git push <remote> :<branch> (since Git v1.5.0)
or
git push <remote> --delete <branch> (since Git v1.7.0)
```

Publish your tags:

```
$ git push --tags
```

Merge & Rebase

Merge into your current HEAD:

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>:

Don't rebase published commit!

```
$ git rebase <branch>
```

Abort a rebase:

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts:

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts:

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved:

```
$ git add <resolved-file>
```

```
$ git rm <resolved-file>
```

Squashing commits:

```
$ git rebase -i <commit-just-before-first>
```

Now replace this,

```
pick <commit_id>
pick <commit_id2>
pick <commit_id3>
```

to this,

```
pick <commit_id>
squash <commit_id2>
squash <commit_id3>
```

Undo

Discard all local changes in your working directory:

```
$ git reset --hard HEAD
```

Get all the files out of the staging area(i.e. undo the last `git add`):

```
$ git reset HEAD
```

Discard local changes in a specific file:

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes):

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit and discard all changes since then:

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

Reset your HEAD pointer to a remote branch current state.

```
git reset --hard <remote/branch> e.g., upstream/master, origin/my-feature
```

Reset your HEAD pointer to a previous commit and preserve all changes as unstaged changes:

```
$ git reset <commit>
```

Reset your HEAD pointer to a previous commit and preserve uncommitted local changes:

```
$ git reset --keep <commit>
```

Remove files that were accidentally committed before they were added to .gitignore

```
$ git rm -r --cached .
$ git add .
$ git commit -m "remove xyz file"
```

Git-Flow

Index

Setup

You need a working git installation as prerequisite. Git flow works on OSX, Linux and Windows.

OSX Homebrew:

```
$ brew install git-flow
```

OSX Macports:

```
$ port install git-flow
```

Linux (Debian-based):

```
$ apt-get install git-flow
```

Windows (Cygwin):

You need wget and util-linux to install git-flow.

```
$ wget -q -O - --no-check-certificate
https://github.com/nvie/gitflow/raw/develop/contrib/gitflow-installer.sh | bash
```

Getting Started

Git flow needs to be initialized in order to customize your project setup. Start using git-flow by initializing it inside an existing git repository:

Initialize:

You'll have to answer a few questions regarding the naming conventions for your branches. It's recommended to use the default values.

```
git flow init
```

Features

Develop new features for upcoming releases. Typically exist in developers repos only.

Start a new feature:

This action creates a new feature branch based on 'develop' and switches to it.

```
git flow feature start MYFEATURE
```

Finish up a feature:

Finish the development of a feature. This action performs the following:

1)Merged MYFEATURE into 'develop'.

2)Removes the feature branch.

3)Switches back to 'develop' branch

```
git flow feature finish MYFEATURE
```

Publish a feature:

Are you developing a feature in collaboration? Publish a feature to the remote server so it can be used by other users.

```
git flow feature publish MYFEATURE
```

Getting a published feature:

Get a feature published by another user.

```
git flow feature pull origin MYFEATURE
```

Tracking a origin feature:

You can track a feature on origin by using

```
git flow feature track MYFEATURE
```

Make a Release

Support preparation of a new production release. Allow for minor bug fixes and preparing meta-data for a release

Start a release:

To start a release, use the git flow release command. It creates a release branch created from the 'develop' branch. You can optionally supply a [BASE] commit sha-1 hash to start the release from. The commit must be on the 'develop' branch.

```
git flow release start RELEASE [BASE]
```

It's wise to publish the release branch after creating it to allow release commits by other developers. Do it similar to feature publishing with the command:

```
git flow release publish RELEASE
```

(You can track a remote release with the: **`git flow release track RELEASE`** command)

Finish up a release:

Finishing a release is one of the big steps in git branching. It performs several actions:

1)Merges the release branch back into 'master'

2)Tags the release with its name

3)Back-merges the release into 'develop'

4)Removes the release branch

```
git flow release finish RELEASE
```

Don't forget to push your tags with **`git push --tags`**

Hotfixes

Hotfixes arise from the necessity to act immediately upon an undesired state of a live production version. May be branched off from the corresponding tag on the master branch that marks the production version.

Git flow hotfix start:

Like the other git flow commands, a hotfix is started with

```
$ git flow hotfix start VERSION [BASENAME]
```

The version argument hereby marks the new hotfix release name. Optionally you can specify a basename to start from.

Finish a hotfix:

By finishing a hotfix it gets merged back into develop and master. Additionally the master merge is tagged with the hotfix version

```
git flow hotfix finish VERSION
```
