Git Cheat Sheet

Table of contents

The basics

Git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with lightning fast performance. It outclasses SCM tools like Subversion, CVS, Perforce, and ClearCase with features like cheap local branching, convenient staging areas, and multiple workflows.

Installation

Install git client wherever you need to run.

Windows	https://windows.github.com
Mac	https://mac.github.com
Fedora	$yum install git (up to Fedora 21) $dnf install git (Fedora 22 and newer)
Debian/Ubuntu	$apt-get install git
Git for all platforms	http://git-scm.com

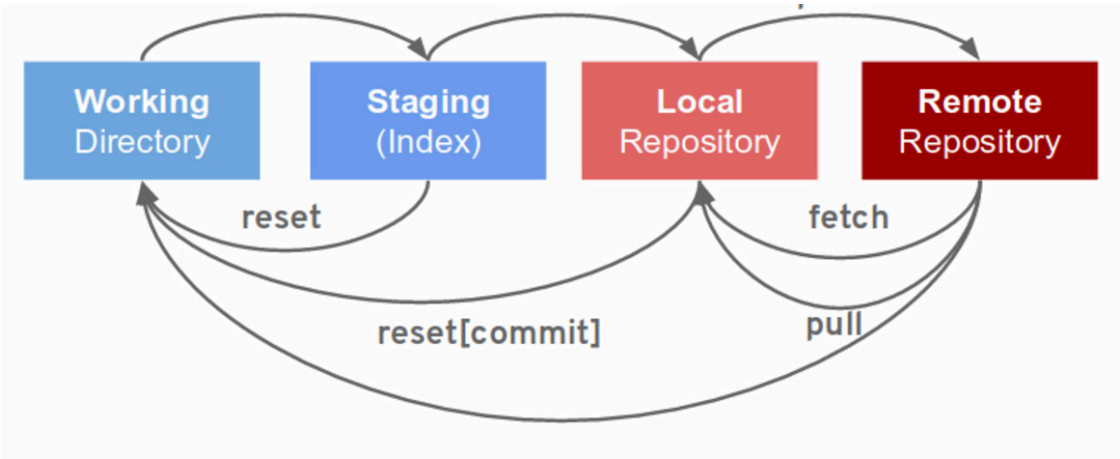
Next, configure user information used across all local repositories.

```
# set a name that is identifiable for credit when reviewing 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
```

Git commands sequence



Git CLI

Create

```
# Clone an existing repository
$ git clone https://github.com/bparees/openshift-jee-sample.git

# Create a new local repository
$ git init
```

Local changes

```
# Changed files in your 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 history

```
# Show all commits, starting with newest
$ git log

# Show changes over time for a specific file
$ git log -p <file>

# Who changed what and when in <file>
$ git blame <file>
```

Branches & tags

```
# List all existing branches
$ git branch -av

# Switch HEAD branch
$ git checkout <branch>

# Create a new branch based on your current HEAD
$ git branch <new-branch>

# Delete a local branch
$ git branch -d <branch>

# Mark the current commit with a tag
$ git tag <tag-name>
```

Update & publish

```
# List all currently configured remotes
$ git remote -v

# Add new remote repository, named <remote>
$ git remote add <shortname> <url>

# Download all changes from <remote>, but don‘t integrate into HEAD
$ git fetch <remote>

# Download changes and directly merge/integrate into HEAD
$ git pull <remote> <branch>

# Publish local changes on a remote
$ git push <remote> <branch>
```

Merge & rebase

```
# Merge <branch> into your current HEAD
$ git merge <branch>

# Rebase your current HEAD onto <branch>
$ git rebase <branch>

# Abort a rebase
$ git rebase --abort

# 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>
```

Undo

```
# Discard all local changes in your working directory
$ git reset --hard 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 previous commit and preserve all changes as
unstaged changes
$ git reset <commit>
```

Quickstart using Git

1. Start a new app project or work on an existing app project

```
# The “git init” command in a specific folder of your new project creates a
new and empty Git repository
$ git init

# The “git clone” command is used to download an existing repository from a
remote server.
$ git clone <remote-url>
```

2. Do your stuff on files

```
# Update your codes or add new files or delete some of them via  your
favorite editor
$ vi hellowWorld.java
…
Change your codes or contents
…
Save and Quit <wq>

# (Optional) Rebase your current HEAD onto <branch>
$ git rebase <branch>
```

3. Keep looking over your work

```
# The “git status” command tells you what happened since the last commit
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
    modified:   ansible/vars.yml
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    ansible/helloworld-msa.retry
no changes added to commit (use "git add" and/or "git commit -a")
```

4. Add files to your staging

```
# The “git add” command enables you to add all changed files to your
staging area
$ git add .

# If you need to add a specific file to your staging, you have to describe
the file name explicitly
$ git add <filename>
```

5. Commit all staged changes

```
# A commit wraps up all the changes you previously staged with the “git
  add” command. To record this set of changes in Git’s database, you
  execute the “git commit” command with a short and informative message
$ git commit -m "message"
```

6. Confirm your work

```
# Running the “git status” command right after a commit proves to you: only
the changes that you added to the Staging Area were committed
$ git status
```

7. Inspect the commit history

```
# The “git log” command lists all the commits that were saved in
chronological order. This allows you to see which changes were made in
detail and helps you comprehend how the project evolved
$ git log
commit 6a8110589c25eac8acd7223d2bf91995c0b72db8
Author: Daniel Oh <doh@redhat.com>
Date:   Fri Mar 24 13:27:03 2017 -0400
  Update Zipkin to 0.1.9
commit 40380a55163a7e93366c01e37d1d0ad5a3afc848
Author: Daniel Oh <doh@redhat.com>
Date:   Thu Mar 23 21:44:09 2017 -0400
  Remove the need to adjust SCC
# Publish local changes on a remote
$ git push
```