# GIT FOR SUBVERSION USERS

presented by TOWER — the best Git client for Mac and Windows

## Creating a New Repository

With `git init`, an **empty** repository is created in the current folder of your local hard drive. The `git add` command then marks the current contents of your project directory for the next (and in this case: first) commit.

SUBVERSION

```
$ svnadmin create /path/to/repo
$ svn import /path/to/local/project http://
    example.com/svn/ trunk -m "Initial import"
```

GIT

```
$ git init
$ git add .
$ git commit -m "Initial commit"
```

## Cloning a Remote Repository

Getting a copy of the project from a remote server seems almost identical. However, after performing `git clone`, you have a **full-blown local repository** on your machine, not just a working copy.

SUBVERSION

```
$ svn checkout
    svn+ssh://svn@example.com/svn/trunk
```

GIT

```
$ git clone
    ssh://git@example.com/path/to/git-repo.git
```

## Inspecting History

To inspect historic commits, both systems use the `log` command. Keep in mind, however, that `git log` doesn't need to ask the remote server for data: your project's history is already at hand, saved in your local repository.

SUBVERSION

```
$ svn log | less
```

GIT

```
$ git log
```

## Committing Local Changes

Inspecting your current local changes is very similar in both systems.

SUBVERSION

```
$ svn status
$ svn diff | less
```

GIT

```
$ git status
$ git diff
```

In case you've created new files or deleted old ones, you should tell Git with the `git add` and `git rm` commands. You'll be pleased to hear that it's safe to inform Git **after** deleting or moving a file or even a folder. This means you should feel free to delete or move even complete directory structures in your favorite editor, IDE, or file browser and **later** confirm the action with the `add` and `rm` commands.

SUBVERSION

```
$ svn add <file>
$ svn rm <file>
```

GIT

```
$ git add <file>
$ git rm <file>
```

In its simplest form, committing can feel just like in Subversion. With the `-a` option, you tell Git to simply **add** all current local changes to the commit.

SUBVERSION

```
$ svn commit -m "message"
```

GIT

```
$ git commit -a -m "message"
```

Although short-circuiting Git's staging area like this can make sense, you'll quickly begin to love it once you understand how valuable it is:

You can add selected files to the staging area and even limit this to certain parts (or even lines) of a file by specifying the `-p` option. This allows you to craft your commits in a very **granular** way and only add changes that belong to the **same topic** in a single commit.

GIT

```
$ git add <file1> <file2>
$ git add -p <file3>
```

## Branching & Tagging

In contrast to Subversion, Git doesn't use directories to manage branches. Instead, it uses a more powerful and lightweight approach. As you might have already noticed, the `git status` command also informs you about which branch you are currently working on. And in Git, you are **always** working on a branch!

SVN
```
$ svn copy http://example.com/svn/trunk/
  http://example.com/svn/branches/<new-branch>
```

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

To switch to a different branch and make it active (then also referred to as the **HEAD** branch), the `git checkout` command is used. Because switching can take some time in Subversion, it's not unusual to instead have multiple working copies on your disk. In Git, this would be extremely uncommon: since operations are very fast, you only keep a single local repository on your disk.

SUBVERSION
```
$ svn switch
  http://example.com/svn/branches/<branch>
```

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

Listing all available local branches just requires the `git branch` command without further arguments.

SVN
```
$ svn list http://example.com/svn/branches/
```

GIT
```
$ git branch
```

Creating tags is just as quick & cheap as creating branches.

SVN
```
$ svn copy http://example.com/svn/trunk/
  http://example.com/svn/tags/<tag-name>
```

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

## Merging Changes

Like in newer versions of SVN, you only need to provide the branch you want to integrate to the `git merge` command.

SUBVERSION
```
$ svn merge -r REV1:REV2
  http://example.com/svn/branches/<other-branch>
$ svn merge                                (or in newer SVN versions)
  http://example.com/svn/branches/<other-branch>
```

GIT
```
$ git merge <other-branch>
```

Everything else is taken care of for you: you can merge two branches as often as you like, don't have to specify any revisions and can expect the operation to be blazingly fast if you're merging two local branches.

If a merge conflict should occur, Git will already update the rest of the working copy to the new state. After resolving a conflicted file, you can mark it using the `git add` command.

SUBVERSION
```
$ svn resolved <file>
```

GIT
```
$ git add <file>
```

## Sharing & Collaborating

To download & integrate new changes from a remote server, you use the `git pull` command.

SUBVERSION
```
$ svn update
```

GIT
```
$ git pull
```

If you only want to download & inspect remote changes (before integrating them), you can use `git fetch`. Later, you can integrate the downloaded changes via `git merge`.

GIT
```
$ git fetch
```

In Subversion, data is automatically uploaded to the central server when committing it. In Git, however, this is a separate step. This means you can decide for yourself **if and when** you want to share your work. Once you're ready, the `git push` command will upload the changes from your currently active branch to the remote branch you specify.

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

Your teammates, too, will publish their work like this on a remote (with the `git push` command). If you want to start working on such a branch, you need to create your own local copy of it. You can use the `git checkout` command with the `--track` option to do just that: create a local version of the specified remote branch. You can later share the additional commits you've made at any time with the `git push` command, again.

SUBVERSION
```
$ svn switch
  http://example.com/svn/branches/<branch>
```

GIT
```
$ git checkout --track <remote>/<branch>
```