- Git_Guide

This page will document git usage making use of newer concepts introduced in git 1.5, for older git versions, see Git_Guide_Pre-1.5.

Sommaire

Intro

For given commands you can type either eg `git commmit` or `git-commit`; the man page is available via `man git-commit` or 'git commit --help'.

For more common everyday commands see the list in `git help`.

This guide assumes that the repository you work with was cloned by git 1.5, if it was cloned with an older version, some of the commands, especially related to branch handling don't work as described here. Either re-clone or use the commands from Git_Guide_Pre-1.5.

Setup

How do I get my copy of the repository?

Once you have your access setup:

```
$ git clone ssh://[<user@>]scm.sourcemage.org/smgl/grimoire.git
$ ls
grimoire
$ cd grimoire
```

Note: If you don't want to authenticate for every action, you can also use git:// instead of ssh:// for read/update operations. You will need ssh:// for any write operations. Do not use http://, even if it seems to work. It is not a supported mechanism for developers getting timely updates.

Any other initial setup I need?

Tell git who you are:

```
$ git config user.name "FirstName LastName"
$ git config user.email "user@example.com"
```

If you have many git repositories under your current user, you can set this for all of them

```
$ git config --global user.name "FirstName LastName"
$ git config --global user.email "user@example.com"
```

If you want pretty colors, you can setup the following for branch, status, and diff commands:

```
$ git config --global color.branch "auto"
$ git config --global color.status "auto"
$ git config --global color.diff "auto"
```

Or, to turn all color options on (with git 1.5.5+), use:

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

To enable aut-detection for number of threads to use (good for multi-CPU or multi-core computers) for packing repositories, use:

```
$ git config --global pack.threads "0"
```

To disable the rename detection limit (which is set "pretty low" according to Linus, "just to not cause problems for people who have less memory in their machines than kernel developers tend to have"), use:

```
$ git config --global   diff.renamelimit "0"
```

Basics

For just dealing with the primary branch of the grimoire (test for the grimoire, aka "master" in git):

How do I keep my copy current?

Assuming you are on the master branch:

```
$ git pull
```

How do I update a spell, assuming I have no other changes I don't want to commit?

```
$ git pull
$ $EDITOR path/to/spell/files
$ git add path/to/new/files
  Note: deletions will be noted automatically.
$ git commit -a
  Note: fill the commit message with the spell name and the version number it is
updated to.
$ git push origin master
```

Note: This commits *all* local changes, if you want to keep some not committed, see the Intermediate Usage section.

Oh no, a conflict! How do I resolve it?

```
$ $EDITOR path/to/conflicting/file
$ git update-index path/to/conflicting/file
$ git commit -F .msg
$ rm .msg
```

I screwed up, how do I reset my checkout?

```
$ git checkout -f
```

I want to fix my last commit.

```
$ $EDITOR path/to/spell/files
$ git commit --amend <some/path>
```

This will include the new changes you made and recommit with the previous commit message.

What other info commands are there?

```
$ git status
$ git log [<some/path>]
$ git show [<some commit id>]
$ git diff
```

Note:

- `git-config color.diff auto` tells git you prefer colored output.

Intermediate Usage

I only want to commit certain files.

```
$ git commit path/to/some/files
```

Note:

- `git commit` with no args may complain about needing to update the index to include some files in the commit; specifying explicit paths will avoid this.
- Best practice is to always do a commit per atomic change so that a separate commit id is available for future cherry-picks, reverts, etc. In other words, if you have 5 spells to update, do a commit per spell directory and not one commit that includes all 5 updates.

I want to discard my uncommitted changes

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

That will discard all the uncommitted changes you made locally. If you want to reset only a part of the repository, you'll have to use a trick:

```
$ git diff some/path | patch -p1 -R
```

A simpler method for discarding changes to part of the repository is:

```
$ git checkout some/path
```

How do I see all the available branches?

```
$ git branch
* master
```

The * denotes the current working branch. "master" is the branch each repository starts with, it is a local branch of the remote "master".

`git branch -r` shows remote branches, and `git branch -a` shows all:

```
$ git branch -r
  origin/HEAD
  origin/devel
  origin/devel-iceweasel
  origin/devel-shadow
  origin/master
  origin/stable-0.3
  origin/stable-0.4
  origin/stable-0.6
  origin/stable-0.7
  origin/stable-rc-0.4
  origin/stable-rc-0.5
  origin/stable-rc-0.6
  origin/stable-rc-0.7
  origin/stable-rc-0.8
```

I want to work on an upstream branch other than "master".

```
$ git checkout --track -b <local name> origin/<remote name>
Note: this creates a local branch <local name> based on the upstream branch and
switches your working copy to that branch.
$ git pull
$ git push origin <local name>
```

Sometimes you may need to use the following instead of that last commit:

```
$ git push origin <local name>:<remote name without origin/>
```

I want to get back to the main branch.

```
$ git checkout master
```

I want to create a local WIP branch.

```
$ git branch devel-something <base branch>
$ git checkout devel-something
edit/add/commit/etc.
```

Or just:

```
$ git checkout -b devel-something <base branch>
edit/add/commit/etc.
```

If <base branch> is currently checked out, you don't have to specify it explicitly. You can use the "--track" argument to branch/checkout to make "git pull" work in this WIP branch without parameters.

I want to push my WIP branch changes back to the mainline.

```
$ git checkout master
$ git pull . devel-something
```

I want to delete a local WIP branch.

```
$ git branch -d devel-something
```

git will refuse to delete the branch if it contains changes that haven't been merged to master yet. To delete the branch anyway, use

```
$ git branch -D devel-something
```

I want to cherry-pick a change from one local branch to another local branch.

```
$ git log <first local branch> path/to/changed/file
$ git checkout <second local branch>
$ git cherry-pick -x <sha1 refspec (commit id) from the previous 'git log'
output>
```

I want to cherry-pick something to a remote branch.

```
$ git checkout --track -b <tmp local branch> origin/<remote branch>
$ git cherry-pick -x <sha1 refspec of commit from other (local or remote)
branch>
$ git push origin <tmp local branch>
$ git branch -D <tmp local branch>
```

I want to revert a commit.

```
$ git log path/to/file
$ git revert <sha1 refspec of the bad commit>
```

Note: `revert` and `cherry-pick` automatically do a commit with a message that preserves the original commit id. `man git-cherry-pick` for options to change this behaviour.

Advanced Usage

I want to publish my WIP branch on the central server.

```
$ git push origin devel-something:refs/heads/devel-something
```

Note:

- Other people updating their clones will automatically get this branch in their remote branches list.
- Branch names have to be unique, so "devel-something" must not yet exist as branch in the remote repository.
- You can remove the branch names from your remote branch listing by using "git branch -d -r origin/<remote branch name>".
- If you do want to delete the remote branch you can use: "git push origin :branch_to_delete".

I want to work in various remote branches often.

Just create a local branch for each remote branch you want to work on:

```
$ git branch --track devel origin/devel
$ git branch --track stable-rc-0.10 origin/stable-rc-0.10
```

I only want to push my commits up to a certain point.

```
$ git push origin <sha1 spec>:<remote branch dst>
```

I'm tired of typing my SSH key passphrase.

You may want to use `ssh-agent` to avoid having to type in your SSH password while working on the repositories (assuming you cloned with developer access). First we load the `ssh-agent` :

```
$ eval `ssh-agent`
Agent pid 10044
```

The above PID will be different for you. Now we load our key (assuming you only have one key or want to load them all):

```
$ ssh-add
```

If you have a specific key for the grimoire and want to only load that, use:

```
$ ssh-add ~/.ssh/grimoire_id_rsa
```

Assuming you named your grimoire SSH key `~/.ssh/grimoire_id_rsa`.

You can also consider using a passphrase-less key, as long as the physical security of the private key file is reasonable (your development machine isn't share with other users, you have firewall protection in place, etc.).

Why do my commit emails have extra "Merge branch 'master'" messages?

When you do a `git pull`, it fetches any changes to the upstream branches, merges them into your checkout, and does an auto-commit back to your local repo. This auto-commit produces this message, and it gets sent along when you push. Note that this commit only happens if you have commits in your local repository that haven't been pushed yet.

If you want to avoid this, you can "rebase" your branch instead of using "pull". Rebasing works by rewinding your working copy to the state of the remote branch, then replays all your changes on top of that.

```
$ git fetch
$ git rebase remotes/origin/<remote branch>
```

Obviously this can lead to conflicts similar to "git pull", but unlike pull, rebase will stop on each conflict. After resolving this conflict (editing files and running "git update-index" on them), "git rebase --continue" will continue the rebase. If you want to skip the conflicting commit, use "git rebase --skip". The whole rebase can be stopped and your branch restored to how it was by using "git rebase --abort".

You can also use "git pull --rebase" and you will avoid this message. This will do rebase automaticly. And there will be no more "Merge branch 'master'" in the log.

I have totally screwed up my repository with a broken merge

Most actions done on your git repository can be undone.

```
$ git reflog
78c80b7... HEAD@{0}: pull : Fast forward
c516234... HEAD@{1}: checkout: moving to master
c516234... HEAD@{2}: pull : Fast forward
01fdb2a... HEAD@{3}: rebase: mplayer: adapt configure options to latest svn
da8ed38... HEAD@{4}: rebase
...
```

This shows all actions that have been done in your repository. If you want to undo that latest pull, you can use "git reset":

```
$ git reset --hard HEAD@{1}
```

Additional resources

- More detailed examples and information on git: Git Magic
- Useful aliases and initial configuration example: Wincent: Git Quickstart
- Good tutorial on the basics of git: A tour of git: the basics
- Simple workflow example illustrating the use of git and quill by JakaKranjc: Simple Workflow Example

- Take a look at this page explaining typical mistakes a Grimoire Guru can but ***must not*** make Common Mistakes
- A few notes on Grimoire Changelog
- The Git Community Book

Troubleshooting

- **I can't push my commits.** You probably don't have repository access. Bug the grimoire lead about it and they should give you access if you should have it.