Git Cheat Sheet

1 Settings

- show git configuration:

```
git config --list
git config --get user.name
```

- set configuration:

```
git config --global --add user.name "John Doe"
git config --add color.ui "auto"
```

`--global` makes the setting global for all repos.

- some common settings:

Setting	**Meaning**
user.name	User name.
user.email	User email.
color.ui	Use colors (“auto”!)?
core.editor	Which editor to use?

2 Basic Git

- Create a repository:

```
cd projDir
git init
```

- Add files to version control:

```
git add file1 file2
```

or

```
git add *
```

If the files have been added before, they will be included in the 'staging area' and thus committed withe the next `git commit`.

- Status, log and information:

  ```
  git status
  git log
  git show [object]
  ```

  `[object]` may be a commit, branch or something like `stash@{0}`.

- Commit changes:

  ```
  git commit changedFile -m "Commit message."
  ```

  or

  ```
  git commit -a
  ```

  Open the editor specified by `core.editor` for editing the commit message and then commit all changed files (skip staging files).

  ```
  git commit file1 file2
  ```

  Only commit `file1` and `file2`, open editor for editing the commit message.

- change last commit:

  ```
  git commit --amend
  ```

  Opens the text editor to change to commit message. Also notices files that have been changed and staged (`git add file`) or removed.

- remove file from version control:
  - also remove file from disk:

    ```
    git rm file
    ```

  - keep file on disk:

    ```
    git rm --cached file
    ```

- go back to `fileName`'s last committed version:

  ```
  git checkout -- fileName
  ```

- get help:

  ```
  git stash --help
  ```

  shows the man page for `git stash`.

- rename a versioned file:

  ```
  git mv oldName newName
  ```

- `diff` for all files:

```
git diff
```

`diff` for a single file:

```
git diff fileName
```

`diff` for changes that are already staged:

```
git diff --cached
```

- let git ignore certain files: create a file `.gitignore` and add it to the repo:

```
# comment
.so
!bla.so
TODO
```

This makes `git` ignore the file `TODO` and all `.so` files, except `bla.so`.

3 Branches

- list branches:

```
git branch
```

Add `-r` for remote branches, use `-a` for remote and local branches.

- create new branch:

```
git branch newBranch
```

Create a branch and check it out immediately:

```
git checkout -b newBranch
```

- change to a branch:

```
git checkout branchName
```

- delete branch:

```
git branch -d branchName
```

for branches that branch off `HEAD`;

```
git branch -D branchName
```

for any branch.

- merge `other` branch into current branch:

```
git merge other
```

- push all branches to remote repository:

```
git push --all
```

- rename a branch:

  ```
  git branch -m oldBranch newBranch
  ```

- checkout single files from another branch to current branch:

  ```
  git checkout branchToUse fileName
  ```

- create a tracking branch (automatically pull and push from/to the tracked branch - used to follow remote changes) `branchName`:

  ```
  git checkout --track remoteAlias/branchName
  ```

  A different local name `localName` can be used with

  ```
  git checkout -b localName remoteAlias/branchName
  ```

  Alternatively,

  ```
  git pull theirBranach
  ```

  will fetch `'origin/theirBranch` and merge with the local `theirBranch` branch.

4 Git Notions

- `HEAD`: pointer the branch we are on.
- `branch`: pointer to a commit.
- `commit`: snapshot of the `git` 'filesystem' including information on parent commits/snapshots.
- `working directory`: copies of files under version control.
- `staging area`: copy of the `git` 'filesystem' to be included in the next `commit`.

5 Using `git` with remote repositories

- add alias `myRepo` for remote repository:

  ```
  git remote add remoteAlias ssh://user@host.domain.tld/directory/myRepo
  ```

- show aliases for remote repositories:

  ```
  git remote
  git remote show remoteAlias
  ```

  The second line gives details (also on branches).

- rename a remote:

  ```
  git remote rename oldAlias newAlias
  ```

- remove a remote (and all tracking branches already fetched):

  ```
  git remote rm remoteAlias
  ```

- clone a copy of a remote repository and create a local repository with a suitable remote `origin` set:

  ```
  git clone URL
  ```

  `clone` will get create a subfolder, fill (fetch) the subfolder with the contents of the repo and then create and checkout the default branch.

- retrieve all remote branches with

  ```
  git fetch remoteAlias
  ```

  No local branches will be altered (merging possibly needed).

- get a specific branch from the remote and start working in it:

  ```
  git checkout -b branchName origin/branchName
  ```

- fetch a remote branch and merge it with the current branch:

  ```
  git pull remoteAlias branchName
  ```

  The working copy shall be clean for this operation.

- after a branch has been deleted from a remote repo,

  ```
  git prune remoteAlias
  ```

  will delete the remote-tracking branches that do not exist in the remote anymore.

- push local changes back to the remote with

  ```
  git push remoteAlias branchName
  ```

  A different name for the branch will be used by

  ```
  git push remoteAlias localBranchName:remoteBranchName
  ```

- delete remote branch:

  ```
  git push remoteAlias :branchName
  ```

5.1 With central repository

- Create a repository on central server:

  ```
  mkdir foo
  cd foo
  git init --bare --shared foo.git
  chgrp -R dev foo.git  (optional)
  ```

  `shared` makes the repo group writable. `bare` means there is no working copy.

- push local repo to server:

  ```
  cd localRepo
  git push ssh://user@host.domain.tld/home/user/foo.git '*:*'
  ```

  (this pushes the local repo with everything to the server)

- clone new working directory that tracks the one on the server:

  ```
  git clone ssh://user@host.domain.tld/home/user/foo.git newRepo
  ```

- after hacking in `newRepo`, update repo on server:

  ```
  cd newRepo
  git push
  ```

  For more options, see above.

5.2 With GitHub

- create repository `repoName` from the web interface

- teach local repository about the remote one:

  ```
  cd repoName
  git remote add origin git@github.com:githubuser/repoName.git
  ```

- push files to GitHub:

  ```
  cd repoName
  git push
  ```

- to clone the GitHub repo:

  ```
  git clone git@github.com:githubuser/repoName.git newRepo
  ```

- push changes back to GitHub:

  ```
  cd repoName
  git push
  ```

  For more options, see above.

6 Discarding changes in working copy

There are at least two different ways to reset to working directory to the last versioned status:

6.1 Checkout: Forget about changes

```
git checkout -- fileName
```

resets `fileName` to the last checked in version - changes in the working directory are lost!

```
git checkout commitName
```

gets back to commit `commitName`. Note that information on `HEAD` is lost in this case. However, `git reflog` still remembers where `HEAD` was.

6.2 Stashes: keep changes

- changes in a working dirctory may be 'stashed' away:

  ```
  git stash save "Status before going back"
  ```

- stashes are listed with:

  ```
  git stash list
  ```

- apply the stash on top of the stack again:

  ```
  git stash apply
  ```

  keeps to stash saved, whereas

  ```
  git stash pop
  ```

  applies the stash and also removes the stash form the list.

- delete a stash:

  ```
  git stash drop
  ```

  deletes the stash on top of the stack, whereas

  ```
  git stash drop stash@{0}
  ```

  deletes the stash `stash@{0}`.

7 Graphical tools

- `git gui`: Perform adding, commiting, branching etc. graphically.
- `gitk`: View commit history and branches.
- `git difftool`: View diffs graphically (needs setting `diff.tool`).

8 Links

- Git reference: http://gitref.org/
- "Pro Git" book: http://progit.org/
- Git community book: http://book.git-scm.com/
- Git with central sever: http://toroid.org/ams/git-central-repo-howto

9 TODO

- info on merging
- learn rebasing
- fix bugs (that certainly do exist in here)