(pdftex.def) Package pdftex.def Warning: Ignoring void output box

rename a remote: `remote rename <oldrname> <newrname>` – this changes your remote branch names, too.

remove a remote reference/alias: `remote rm <rname>`

Tagging specific points in history

list available tags: `tag [-l] [<pattern>]` – lists all tags in alphabetical order or only those match-ing a pattern.

tag types: A lightweight tag is just a pointer to a specific commit. Annotated tags are stored as full objects in the Git database. They're checksummed; contain the tagger name, e-mai, and date; have a tagging message; and can be GPG-signed.

create a tag: `tag [-a|-s] <tagname> [-m <tag-message>] [<cksum>]` – use `-a` for annotated tags or `-s` for annotated and signed tags. Don't supply `-m`, `-a` or `-s` for lightweight tags. To tag an older commit, apply (part of its) checksum at the end.

show tag information: `show <tagname>`

verify a signed tag: `git tag -v <tagname>`

Tips and Tricks

Shell completion: Take the `git-completion.bash` file from the Git source code and source it in your `.bashrc`. The Windows Git Bash has auto-completion preconfigured.

Command Aliases can be defined via `git config --global alias.<alias> <command>`. The command can be sth. like `checkout`, `'reset HEAD --'` etc.

Branches

A branch is a lightweight movable pointer to a commit. `HEAD` is a special pointer to the branch you're currently

(pdftex.def) Package pdftex.def Warning: Ignoring void output box

on. Usually this is the default branch `master`. The current branch always points to the last commit you made.

create new branch: `branch <bname>` – the newly created branch points to your last commit

switch to another branch: `checkout <bname>` – changes `HEAD` and the files in your working directory appropriately.

create new branch and switch to it: `checkout -b <bname> [<basebranch>]` – You can let start the new branch at another base than the current branch (e.g., a remote branch).

merge branches: First, switch to the branch you want to merge into (the branch that is going to be changed by the merge). Then run `git merge <bname>`. If no conflicts occur, Git automatically creates a new commit object that contains the merged work, and moves the current branch to point on it.

delete an obsolete branch: `branch -d|-D <bname>` – `-D` is a force-deletion for branches with unmerged work.

show and clean merge conflicts: Show conflicts with `git status` and clean up the files manually; then mark them as resolved using `git add`. Alternatively, use `git mergetool`, which fires up an appropriate visual merge tool and walks you through the conflicts. Finally you have to `commit` all these changes.

see last commit on each branch: `git branch -v`

show branches already merged into the current: `branch --merged` – Branches listed without an `*` can be deleted safely.

show all branches with unmerged work: `branch --no-merged` – these branches cannot be deleted with `-d`, you'll have to use `-D`.

access branches on remote sites: use `<rname>/<bname>`. After a `git fetch <rname>`, this will point to the current remote changes.

Git Cheat Sheet

based on **Pro Git** by Scott Chacon: `http://git-scm.com/book`

General Setup with `git config`

Usage `git config [--system|--global] <options>` – operate system-wide, user-specific or just for current project

view settings: `--list | -l`

add a setting: `--add <key> <value>`

delete a setting: `--unset <key>`

setting identity: with `git config ...`

```
... --global user.name "John Doe"
... --global user.email johndoe@example.com
```

git through proxy: `--add http.proxy http://wwwproxy.my.com:8080`

Getting Help

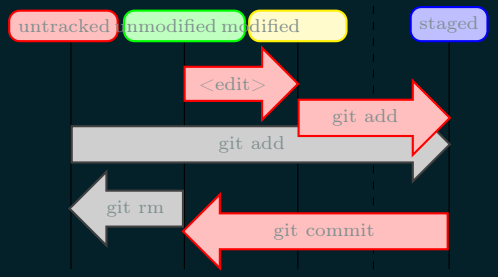
```
git help <verb> | git <verb> --help
man git-<verb>
```

Getting a Git Repository

the leading word `git` will be ommitted for all com-mands.

initialize a new repo in an existing directory: standing in the directory, run `init` and add relevant files, e.g. `add *.c` and `commit -m 'initial project version'`

create a working copy: `clone <url> [<name_of_new_dir>]`; while `<url>` is sth. like `git://github.com/what/ever.git` or `http[s]://...` or `user@server:/path.git`



file status lifecycle in a local repository

(pdftex.def) Package pdftex.def Warning: Ignoring void output box

Recording Changes

check status: `status` – lists all untracked files, modified files (in section "Changed but not updated") and staged files (in section "Changes to be committed"). No output means that all files are unmodified.

track new files: `add <file>` – the file will now be tracked and the current version is going to be staged. If you edit this file again after doing `add` and before doing `commit`, the version of the file at the time you ran `add` is what will be in the histori-cal snapshot. If this is not what you want you have to rerun `add`.

prepare modified files for commit: `add [<file>]` – stages the current version of the given file (or all modified files that are not excluded by `.gitignore`).

protect files from being tracked: or automatically added: Prepare a `.gitignore` file with a content similar to the following:

```
# a comment this is ignored *.a       #
no .a files
! lib.a # but do track lib.a/ TODO # ignore the
root TODO file (↩
    not subdir/TODO)
build/ # ignore all files in the build↩
    / directory
```

```
(pdftex.def)                Package pdftex.def
Warning:  Ignoring void output box
doc/*.txt # ignore doc/notes.txt, but ↩
    not doc/server/arch.txt
```

view modifications in detail: `diff [<file>]` – shows what you've changed but not yet staged (compares working directory vs. staging area). `diff --cached [<file>]` – shows what you've staged (that will go into your next commit).

commit staged changes: `commit [-m "message"]`

commit all changes: `commit -a [-m "message"]` – automatically stages every [already tracked] file, letting you skip the `git add` part.

delete a file: `rm <file>` – deletes the file from working directory and from staging area. You have to `commit` the removal. To keep the file untracked in your working copy, do `rm --cached`

move/rename a file: `git mv <oldfile> <newfile>` works the same way like

```
mv <old> <new>
git rm <old>; git add <new>
```

because Git figures out that this is a rename implicitly.

Filename Globbing: `log/\*.log` matches all files that have the `.log` extension in the `log/` directory. The backslash in front of the `*` is necessary because Git does its own filename expansion.

Viewing the Commit History

`git log` shows commit logs (most recent first) with checksum, author's name and e-mail, date and commit message. The most important options are:

show with diffs: `-p`

show newest *n* commits: `-n` or `-nn`

show abbreviated stats for each commit: `--stat|--shortstat` – prints a list of modified files, how many files were changed, and how many lines were added/removed for each commit. It also puts a summary of the information at the end.

(pdftex.def) Package pdftex.def Warning: Ignoring void output box

pretty log output: `--pretty=<style>` – possible styles are `oneline`, `short`, `full`, `fuller` and `format:"formatstring"` (see help). `--graph` adds a nice little ASCII graph showing your branch and merge history.

limit shown timerange: `--since|--after|--until|--before=<date>` where the date can be specific as in `2008-01-15` or relative as in `2.years.1.day.3.minutes`.

Undoing Things

change last commit: `commit --amend` – takes your staging area and uses it for changing the last commit (useful for adding forgotten files or changing the commit message).

unstage a staged file: `reset HEAD <file>`

revert a modified file: `checkout -- <file>`

Working with Remote Repositories

list all remote places/aliases: `remote [-v]`

add remote repository alias: `remote add <rname> <url>` – adds `<url>` to be accessible via the short name `<rname>`

inspect a remote: `remote show <rname>`

mirror remote changes to local repo copy: `fetch <rname>`

fetch and merge remote to local: `pull <rname>` – this fetches data from the server you originally cloned from and automatically tries to merge it into the code you're currently working on.

push to remote: `push <rname> [<localbranch>[:<remotebranch]] [<tagname>|--tags]`, Only the `master` branch, and branches created from an remote branch, will be pushed automatically. Tags are not pushed by default.