Version Control System

Document Authors: arho.virkki@vtt.fi, anna.hammais@tyks.fi

Repositories

Data flow and analyses

```
Group  Directory                 # Description
-------------------------------------------------------------------------------------
ktp    ETL.git                   # ETL-scripts (with Pentaho Kettle)
ktp    refinery-scripts.git      # Data refinery and enrichments sripts (derived values)
ktp    analyses.git              # Matematical and statistical analyses for automated use
ktp    poiminta.git              # Data subsetting and extraction scripts for R&D use
```

Tools in production

```
Group  Directory                 # Description
-------------------------------------------------------------------------------
ktp    Common.git                # Generic scripts (including automated backups)
ktp    luovutusrekisteri.git     # KTP data extraction log for public www page
ktp    Luovutusprosessit.git     # Data extraction process (for Auria Biobank)
ktp    luovutusrekisteri_java.git # KTP data extraction log
ktp    RtoolsKTP.git             # Generic CCI R utilities
auria  TextMine.git              # Auria text mine scripts
```

Tools in test and prototypes

```
Group  Directory                 # Description
-----------------------------------------------------------------
ktp    backend.git               # Mikko K's UI backend test
ktp    frontend.git              # Mikko K's UI frontend test
ktp    geojson_testing.git       # Mikko K's GeoJSON test
ktp    ktp_interface.git         # Mikko K's old UI repository
ktp    ktp_website.git           # Old KTP web page
ktp    api_dev.git               # Arho's REST-test
```

Using Git

For more information on Git, consult

- Git Cheat Sheet,
- DZone Git Reference Card and
- Git Home Page

Figure 1. The basic Git workflow.

Pre-requisites for every user before using Git

These customizations are done on the clinet machine, e.g. at *ktpanalytics.vsshp.net* or at the machine you happen to be using.

First, set user name and email (such that *git blame* can incriminate you for bugs):

```
git config --global author.name "Arho Virkki"
git config --global user.email "arho.virkki@vtt.fi"
git config --global push.default simple
```

Having colored output in the Terminal application is optional, but can clarify workflows

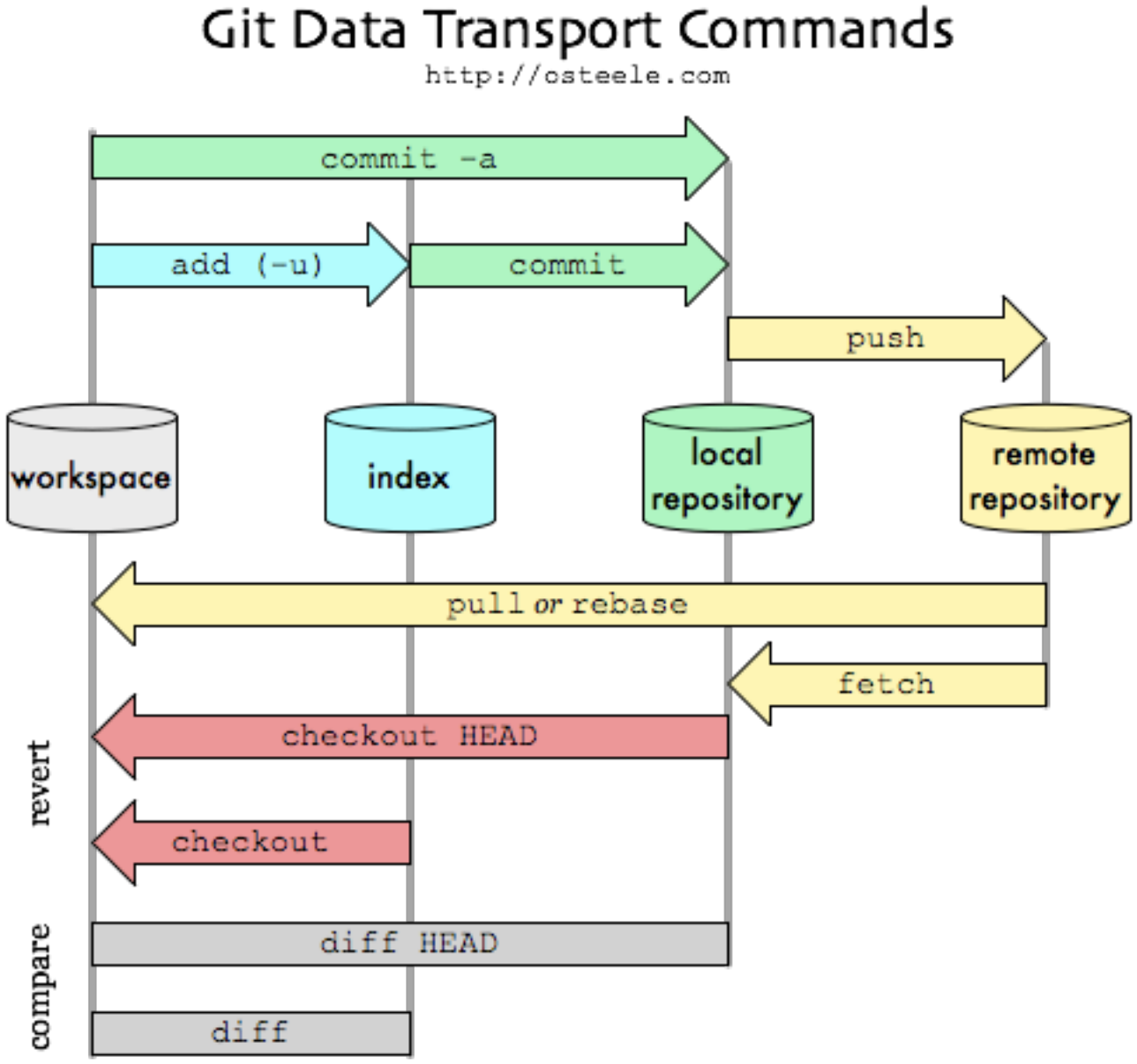
Git Data Transport Commands
http://osteele.com
commit -a
add (-u)
commit
push
workspace
index
local repository
remote repository
pull or rebase
fetch
checkout HEAD
checkout
diff HEAD
diff
revert
compare

Figure 1:

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

For setting custom colors for the output of a specific command, add e.g. the following to the ~/.gitconfig (this sets the colors for the status command)

```
[color "status"]
    added = green bold
    changed = magenta bold
    untracked = cyan bold
```

Git uses *vi* as the default editor. If that is not your preferred choice, choose *nano* instead: * git config –global core.editor “nano”

See that everything is saved in *~/.gitconfig*

```
git config --list
```

Finally, set up ssh keys to simplify usage (since the ssh password does not have to be typed every time). Generate a ssh-key-pair (do not overwrite the old key, if you have already done so)

```
ssh-keygen
```

and issue

```
ssh-copy-id -i ~/.ssh/id_rsa.pub ktp@ktpgit.vsshp.net
```

Creating a new Git repository

Bare Git repositories can be initiated in *shared mode* (*git init –bare –shared*), but in this example, all users access *ktpgit.vsshp.net* with the single user account *ktp*.

Log in to *ktpgit.vsshp.net*

```
ssh ktp@ktpgit.vsshp.net
```

To create a repository called *KTPCommon*, issue

```
mkdir -p /opt/git/Common.git
cd /opt/git/Common.git
git init --bare
```

and log out. That’s all.

In case that multiple users are needed at the *ktpgit.vsshp.net* machine which can access the same repository, create a dedicated group (e.g. git) and set all members of the project as members in that group. When creating the Git repository, *git init –bare –shared* will set the SGID permissions correclty, if the directory ownershis is set correctly.

Using the Newly Created Git Repository

On the local machine, issue

```
git clone ktp@ktpgit.vsshp.net:/opt/git/KTPCommon.git
```

To obtain a copy of the Git repository.

In some situations, Git will not accept the local working directory path on VSSHP M drive, e.g.

```
//vsshp/fs/home/hammaisa/DATA/Git
```

To solve this:

```
cd M:
cd DATA
cd Git
git clone ktp@ktpgit.vsshp.net:/opt/git/KTPCommon.git
```

About Repository Structure

Git repositories can be either live or bare. Bare repositories are initialized with *git init –bare*, with the optional *–shared* argument, they typically reside on the server, and only store the version history. Live repositories are used for the actual work. Bare repositories look like */opt/git/MyRepo.git*, and the corresponding user repository look like */home/user/workspace/MyRepo/*.

Git history

```
git log --oneline
git log --pretty=raw
git log --grep=<pattern from messages>
git log --author=<pattern from author>
git show -s --pretty=raw b8add21
```

Garbage collection

In Git term, garbage collection equals repository cleanup which may save space and speed up processing.

```
git gc
```

Excluding files in Git repositories

To exclude files in git, put them into *.git/info/exclude* where also wildcards and simple regular expressions are allowed. The other option is to create a file *.gitignore* and list the files there. Wildcards are allowed.

To use .gitignore, in terminal, navigate to the location of your Git repository and create a .gitignore file.

Example-1 if you would like to all *.xlsx files to be ignored from Git add following line in to .gitignore

```
*.xlsx
```

Example-2, if you like only .ibynb files to be covered by Git, add following two lines in to .gitignore

```
*.*
!*.ipynb*
```

Branching

- Branching