Using the MaSuRCA software for genome assembly
masurca -g config.txt # Modifying the default parameters in the config.txt configuration file: USE_LINKING_MATES = 1，JF_SIZE = 60000000000

Using QUAST to align the reference genome and obtain non-ref sequences.
quast -t 16 --split-scaffolds --min-alignment 500 --min-identity 90 --unaligned-part-size 500 --min-contig 500 -r Sus_scrofa.Sscrofa11.1.dna.toplevel.fa assembly.fa

Align non-ref sequences to the NT database for contamination removal.
blastn -db nt -query unalign.fasta -out blast.out -evalue 1e-5 -outfmt 7 -max_target_seqs 1 -num_threads 12

Using CD-HIT to cluster and remove redundancy from non-ref sequences.
cd-hit-est -i nonref.fa -o cdhit -c 0.9 -T 80 -n 8 -d 0 -M 0

Annotation of repetitive elements in non-ref sequences.
BuildDatabase -name genome genome.fa
RepeatModeler -database genome -pa 96 -LTRStruct >& run.out
RepeatMasker -pa 120 -species pig -poly -html -gff -dir step1 panseq.fa 1>log1.o.txt 2>log1.e.txt
RepeatMasker -pa 120 -lib repeatmodeler/genome-families.fa -poly -html -gff -dir step2 step1/panseq.fa.masked 1>log2.o.txt 2>log2.e.txt

Annotation of protein-coding genes.
braker.pl --species=pigpan --genome=cat.sm.fa --bam=transcripts.bam --softmasking --cores=40 --gff3 --workingdir=run --addUTR=on --prot_seq=proteins.fa --etpmode
mpiexec -n 120 maker -fix_nucleotides -base round1.out maker_opts.rnd1.ctl maker_bopts.ctl maker_exe.ctl

Functional annotation of genes.
diamond blastp --db uniprot_sprot -q pan.aa --more-sensitive -p 32 -e 1e-6 --max-hsps 1 -k 1 -f 6 -o swissprot.blastp > swissprot.blastp.log 2>&1
diamond blastp --db nr -q pan.aa --more-sensitive -p 64 -e 1e-6 --max-hsps 1 -k 1 -f 6 -o nr.blastp > nr.blastp.log 2>&1
interproscan.sh -b pan -cpu 72 -dp -goterms -i pan.aa -iprlookup -pa
emapper.py --cpu 60 --override -i pan.aa --itype proteins -m diamond --output pan --output_dir ./ --temp_dir ./temp --tax_scope auto --go_evidence non-electronic --target_orthologs all --seed_ortholog_evalue 0.001 --seed_ortholog_score 60 --query_cover 20 --subject_cover 0 --index_chunks 1 --dbmem

Using OrthoFinder for gene family analysis. The "primary_transcripts" folder contains all gene protein sequences.
orthofinder -t 80 -a 8 -f primary_transcripts/

SNP imputation and phasing
java -XX:ParallelGCThreads=4 -Xmx50g -jar beagle.18May20.d20.jar gt=chr.vcf.gz out=impute.chr nthreads=12

calculate chromosome recombination rate
pyrho make_table -n 100 -N 150 --mu 2.5e-8 --logfile make_table.log --outfile lookuptable.hdf --approx --smcpp_file smc.pop_sizes.csv --decimate_rel_tol 0.1 --numthreads 80
pyrho hyperparam -n 100 --mu 2.5e-8 --blockpenalty 100,200,300 --windowsize 50,100,200,400 --logfile hyperparam.log --tablefile lookuptable.hdf --num_sims 10 --smcpp_file smc.pop_sizes.csv --outfile hyperparam_results.txt --numthreads 16
pyrho optimize --tablefile lookuptable.hdf --vcffile chr.vcf.gz --outfile chr.rmap --blockpenalty 100 --windowsize 200 --logfile optimize.log --numthreads 16

Detecting TE insertion polymorphisms (TIPs) using PoPoolationTE2.
generate a ppileup (physical pileup) file
java -jar -Xmx800G popte2.jar ppileup --bam input1.bam [--bam input2.bam ...] --map-qual 15 --hier te-hierachy.txt --output all.ppileup.gz
identify signatures of TE insertions from the ppileup-file
java -jar -Xmx800G popte2.jar identifySignatures --ppileup all.ppileup.gz --mode joint ---output all.signatures --min-count 3 --signature-window fix100 --min-valley fix100
estimate the population frequency of TE insertions and rearrangements
java -jar -Xmx800G popte2.jar frequency --ppileup all.ppileup.gz --signature all.signatures --output all.freqsignatures
filter signatures
java -jar -Xmx800G popte2.jar filterSignatures --input all.signatures --output filtered.signatures --max-otherte-count 2 --max-structvar-count 2 --min-count 0.2
pairs matching signatures of TE insertions
java -jar -Xmx800G popte2.jar pairupSignatures --signature filtered.signatures --ref-genome temerged-reference.fa --hier te-hierachy.txt --output teinsertions.txt

Performing selective sweep analysis using nSL.
selscan --nsl --vcf subpop.vcf --out subpop
norm --nsl --files nsl.out --bp-win --winsize 100000

Performing selective sweep analysis using iHS
selscan --ihs --vcf subpop.vcf --pmap --out subpop --threads 8
norm --ihs --files ihs.out --bp-win --winsize 100000

Performing selective sweep analysis using XPEHH
selscan --xpehh --vcf subpop1.vcf --vcf-ref subpop2.vcf --pmap --out subpop1vs2 --threads 8
norm --xpehh --files xpehh.out --bp-win --winsize 100000

Python script for distance matrix:

```
#!/usr/bin/env python
import re
import sys

def ibsMartrix(infile, outfile):
    dic1 = {}
    lis1 = []
    vls = []
    outfile = open(outfile, 'w')
    for line in open(infile):
        if re.search(r'^\s*FID', line):
            pass
        else:
            info = re.split('\s+', line.strip())
            if info[1] not in lis1:
                lis1.append(info[1])
            if info[3] not in lis1:
                lis1.append(info[3])
            dic1[info[1] + '_' + info[3]] = info[-3]
            dic1[info[3] + '_' + info[1]] = info[-3]
            vls.append(float(info[-3]))

    rg = max(vls) - min(vls)
    maxv = max(vls)

    outfile.write('Sample\t' + '\t'.join(lis1) + '\n')

    for i in lis1:
        outline = [i]
        for j in lis1:
            if i == j:
                outline.append('0')
            else:
                # tv = 1 - float(dic1[i + '_' + j])
                tv = (maxv - float(dic1[i + '_' + j])) / rg
                outline.append(str(tv))
        outfile.write('\t'.join(outline) + '\n')



infile = sys.argv[1]        # pair-wise IBS similarity scores produced by PLINK
outfile = sys.argv[2]       # output file name for distance martrix
ibsMartrix(infile, outfile)
```